凌阳科技

嵌入式微处理器 SPCE3200 原理及应用

V0.8 – 2007-4-25

凌阳单片机技术资料

http://www.unsp.com

版权声明



目　录






凌阳科技

1 S+core7 体系结构

1.1 S+core 简介

1.1.1 简介

S+core 微处理器是采用凌阳指令集架构（Sunplus ISA）的 32 位的 RISC 处理器，该微处理器架构支持 32 位/16 位混合指令模式以及并行条件执行（正在申请专利保护），从而提高了代码密度、性能，使 score 内核得到了广泛的应用。在 S+core 微处理器中采用了 AMBA 总线，为 SOC 集成、扩展协处理器和用户接口提供了灵活性，score 使用 SJTAG 技术使测试和调试程序更加有效。

S+core 微处理器具有如下特征：

- 支持 32 位与 16 位混合指令模式
- 支持并行条件执行
- 提供软件安全设计
- 采用哈佛（Harvard）结构，包含 I-Cache（4K）和 D-Cache（4K）
- 采用 Fixed-MMU（固定映射模式）
- 采用 AMBA 总线规格，可以方便的实现 Soc 集成
- 63 个硬件中断，2 个软件中断，中断采用中断向量
- 采用 SJTAG 协议

1.2 体系结构直接支持的数据类型

S+core 处理器支持以下几种类型的数据：

- 字节（Byte）： 8 位
- 半字（Halfword）： 16 位
- 字（Word）： 32 位

表 1.1 列出了以上各种数据类型对应的数据范围。

表 1.1 数据类型

数据类型		范围	备注
字节	有符号数	$-2^7 \sim +2^7-1$	
	无符号数	$0 \sim +2^8-1$	
半字	有符号数	$-2^{15} \sim +2^{15}-1$	必须半字边界对齐
	无符号数	$0 \sim +2^{16}-1$	
字	有符号数	$-2^{31} \sim +2^{31}-1$	必须字边界对齐

所有的数据处理操作都是以“字”为单位的，例如 ADD 操作。Load/Store 操作可在寄存器和存

储器之间传送字节、半字和字的数据，但是当字节或半字类型的数据被装载时，会自动进行零扩展或符号扩展。

32 位指令正好是一个字，是字（4 字节）边界对齐的，16 位指令正好是一个半字，是半字（2 字节）边界对齐的。

1.3 处理器模式

当前，几乎所有 32 位微处理器在实际应用中都要运行操作系统，为在内核级给操作系统提供支持，微处理器可以在多种模式下运行。

S+core 微处理器支持三种处理器模式：用户模式（User Mode）、核心模式（Kernel Mode）、调试模式（Debug Mode）。

1.3.1 用户模式

用于执行应用程序或操作系统程序。通常情况下，处理器均处于用户模式，直到发生异常，处理器切换到核心模式。处理器处于用户模式时，用户不能访问被系统保护的资源。

1.3.2 核心模式

该模式是操作系统专用的模式。当处理器通过异常进入核心模式后将一直处于该模式，直到一条从异常中返回的指令（RTE，Return From Exception）被执行。

1.3.3 调试模式

该模式用于用户调试阶段。在该模式下，用户程序可以完全访问用户模式和核心模式下的寄存器，以及其它一些调试寄存器。

1.4 内部寄存器

1.4.1 概述

- S+core 处理器包含以下寄存器：
- 32 个通用寄存器（GPR）
- 2 个 Custom Engine 寄存器（CEH、CEL）
- 3 个特殊功能寄存器
  - Sr0：循环计数寄存器（CNT）
  - Sr1：装载合并寄存器（LCR）
  - Sr2：存储合并寄存器（SCR）
- 19 个系统控制寄存器
- 3 个调试控制寄存器

处理器运行在用户模式下，用户程序可以访问 32 个通用寄存器（GPR）、2 个 Custom Engine 寄存器（CEH、CEL）以及 3 个特殊功能寄存器（CNT、LCR、SCR）。处理器运行在核心模式下，用户程序除了可访问所有用户模式下的寄存器外，还可访问 19 个系统控制寄存器。处理器运行在调

试模式下，用户程序除了可访问所有用户模式下以及核心模式下的寄存器外，还可访问 3 个调试模式下寄存器（DSAVE、DEPC、DREG）。各模式下可访问寄存器参考图 1.1 所示：

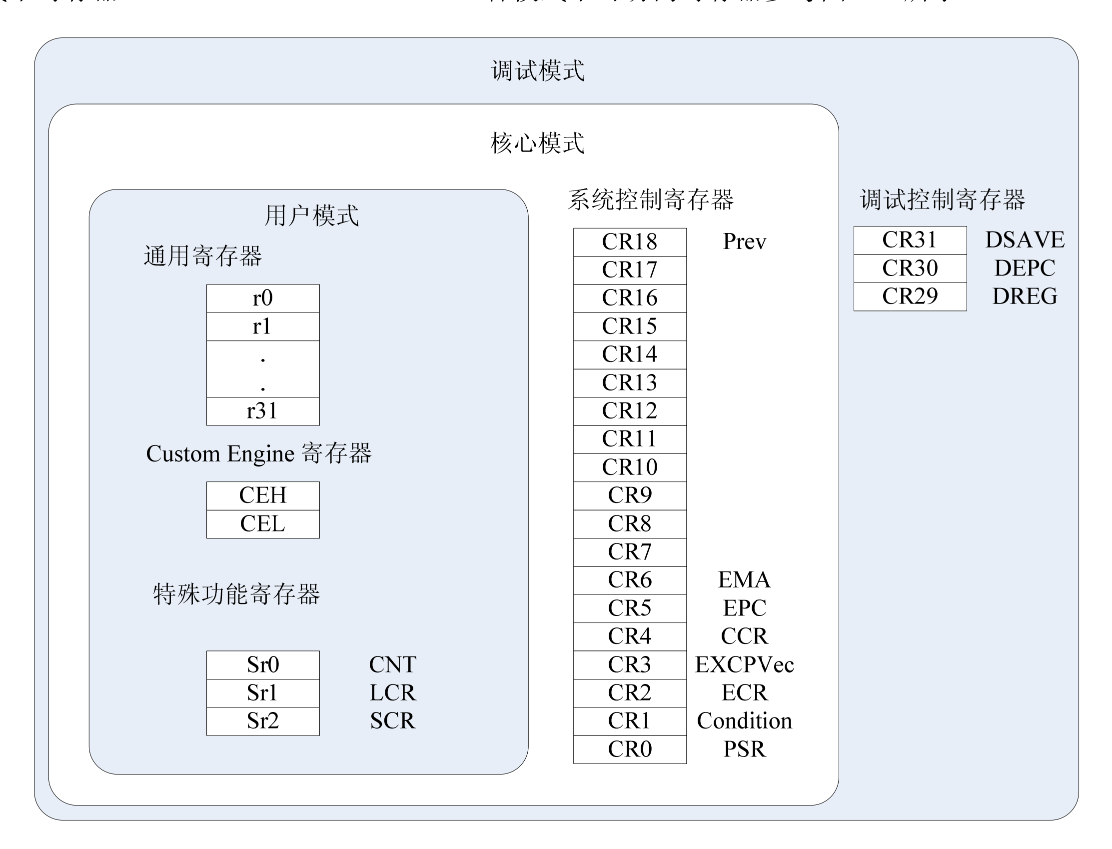


图 1.1 S+core 内核寄存器集

1.4.2 通用寄存器（GPR）

S+core 处理器有 32 个 32 位的通用寄存器（r0~r31）。32 位指令模式下，所有这些通用寄存器均可以被访问。由于指令编码的限制，16 位指令模式下，只有 r0~r15 可以被访问。在跳转/分支或链接指令中，r3 寄存器被用作链接寄存器，用于保存下一条指令地址。如图 1.2 所示：

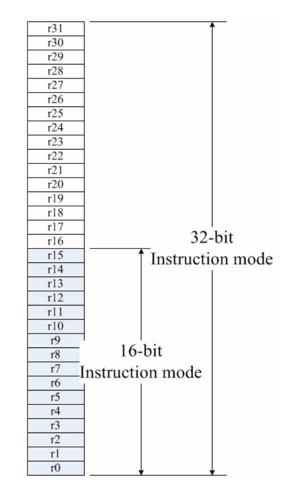


图 1.2 通用寄存器

1.4.3 Custom Engine 寄存器（CEH/CEL）

Custom Engine 寄存器包括 CEH 和 CEL 两个寄存器，用来存储乘法/除法的运算结果。乘法运算完成后，双字的运算结果的高字被放到 CEH 寄存器中，低字被放到 CEL 寄存器中。除法运算完成后，余数放到 CEH 寄存器中，商放到 CEL 寄存器中。

通过 MFCEH、MFCEL、MFCEHL、MTCEH、MTCEL 或 MTCEHL 指令，可实现这两个寄存器与通用寄存器之间的数据传送。详细请参考指令系统一章。

1.4.4 特殊功能寄存器（Srn）

S+core 处理器有 3 个特殊功能寄存器：CNT（Sr0）、LCR（Sr1）、SCR（Sr2）。CNT 寄存器是一个 32 位寄存器，可用来作循环计数。当执行特定的分支指令时，计数减一。例如，执行 bcnz 分支指令时，如果 CNT 寄存器中的值不为零，CNT 寄存器中的值将减 1，程序跳转到目标地址；如果 CNT 寄存器中的值为零，则 bcnz 指令将被视为 nop 指令，CNT 寄存器中的值保持不变。

LCR 寄存器和 SCR 寄存器则是用于存取不对齐的 Load 和 Store 指令操作的。

1.4.5 控制寄存器（CR）

处理器运行在核心模式下有 19 个系统控制寄存器，在调试模式下有 3 个调试控制寄存器。其中有些寄存器没有被定义，参考表 1.2：

表 1.2 控制寄存器

寄存器名称	助记符	寄存器编号
程序状态寄存器	PSR	CR0
条件寄存器	Condition	CR1
异常原因寄存器	ECR	CR2
异常向量寄存器	EXCPVec	CR3
Cache 控制寄存器	CCR	CR4
异常程序计数器	EPC	CR5
异常存储器地址寄存器	EMA	CR6
-	-	-
LIM 物理帧号	LIMPFN	CR15
LDM 物理帧号	LDMPFN	CR16
-	-	-
Prev 寄存器	Prev	CR18
Debug 寄存器	DREG	CR29



寄存器名称	助记符	寄存器编号
Debug 异常程序计数寄存器	DEPC	CR30
Debug 异常内容保存寄存器	DSAVE	CR31

程序状态寄存器（PSR）：

程序状态寄存器用于指示协处理器是否可用，中断屏蔽位、大小端及保存处理器的模式等。

表 1.3 程序状态寄存器（PSR）

位	b31~b29	b28	b27~b24	b23~b18	b17~b16	b15
读/写	R/W	R/W	R/W	R/W	R/W	R
默认值	0	0	0	0	0	1
名称	CU[2:0]	CRA	-	IM_H[5:0]	IM_S[1:0]	Endian
位	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
读/写	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0
名称	UMb	IEb	UMs	IEs	UMc	IEc

CU[2:0]　　b31~b29　　CU[n] = 1：协处理器 n+1 是可用的，协处理器指令可以被执行

CU[n] = 0：协处理器 n+1 是不可用的，协处理器指令不可以被执行

CRA　　b28　　0：控制寄存器在用户模式下是不可访问的，控制寄存器指令在用户模式下不可以被执行

1：控制寄存器在用户模式下是可访问的，控制寄存器指令在用户模式下可以被执行

-　　b27~b24　　保留。读操作时，返回 0；写操作时，只能写 0

IM_H[5:0]①　　b23~b18　　中断屏蔽位（根据中断优先级），对应 63 个硬件中断

IM_S[1:0]①　　b17~b16　　中断屏蔽位（根据中断优先级），对应 2 个软件中断

Endian　　b15　　字节排列方式：

0：小端方式（Little Endian）

1：大端方式（Big Endian）

-　　b14~b6　　保留。读操作时，返回 0；写操作时，只能写 0。

UMb②　　b5　　前一次处理器模式的备份：

0：核心模式

1：用户模式

IEb② b4 前一次中断使能位的备份：

0：63 个硬件中断及 2 个软件中断被禁止

1：63 个硬件中断及 2 个软件中断被允许

UMs② b3 前一次处理器模式：

0：核心模式

1：用户模式

IEs② b2 前一次中断使能位：

0：63 个硬件中断及 2 个软件中断被禁止

1：63 个硬件中断及 2 个软件中断被允许

UMc② b1 当前的处理器模式：

0：核心模式

1：用户模式

IEc② b0 当前的中断使能位：

0：63 个硬件中断及 2 个软件中断被禁止

1：63 个硬件中断及 2 个软件中断被允许

注①：IM[7:2]是一个 6 位的代码，可编码成 0，1，…，63（$2^6 - 1$）的数值。0 表示所有硬件中断请求都被允许。n（1~63）表明优先级为 n 的以及低于 n 的硬件中断请求均被屏蔽。IM[1:0]是软件中断屏蔽位，IM[1]或 IM[0] = 1 表明软件中断 1 或 0 的请求被屏蔽。

注②：UMb、IEb、UMs、IEs、UMc 以及 IEc 六位形成了一个三级的硬件栈，用来保存处理器模式（UM）和中断使能（IE）的信息。UMc 和 IEc 保存当前的 UM 和 IE 值；UMs 和 IEs 保存发生异常前的 UMc 和 IEc 值；UMb 和 IEb 是 UMs 和 IEb 的一个备份。例如，异常发生时，硬件先把当前的 UMc 和 IEc 值保存在 UMs 和 IEs 中，然后再重新设置 UMc 和 IEc；执行异常返回指令 RTE 时，UMs 和 IEs 中的值自动拷贝回 UMc 和 IEc。

条件寄存器（Condition）：

条件寄存器指示当前程序运行是否出现溢出、进位（借位）、零标志、负标志及并行条件执行标志（T），分支跳转指令就是根据这些标志位判断程序的流向。

表 1.4 条件寄存器（Condition）

位	b31~b15	b14	b13	b12	b11	b10	b9	b8
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

默认值	0	0	0	0	0	0	0	0
名称	-	Tb	Nb	Zb	Cb	Vb	Ts	Ns
位	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0	0	0
名称	Zs	Cs	Vs	Tc	Nc	Zc	Cc	Vc

-	b31~b15	保留。读操作时，返回 0；写操作时，只能写 0
Tb	b14	前一次的条件并行执行标志位 T 的备份
Nb	b13	前一次的负标志位 N 的备份
Zb	b12	前一次的零标志位 Z 的备份
Cb	b11	前一次的进位/借位/扩展标志位 C 的备份
Vb	b10	前一次的溢出标志位 V 的备份
Ts	b9	前一次的条件并行执行标志位 T
Ns	b8	前一次的负标志位 N
Zs	b7	前一次的零标志位 Z
Cs	b6	前一次的进位/借位/扩展标志位 C
Vs	b5	前一次的溢出标志位 V
Tc	b4	当前的条件并行执行标志位 T
Nc	b3	当前的负标志位 N
Zc	b2	当前的零标志位 Z
Cc	b1	当前的进位/借位/扩展标志位 C
Vc	b0	当前的溢出标志位

Tb、Nb、Zb、Cb、Vb、Ts、Ns、Zs、Cs、Vs、Tc、Nc、Zc、Cc 以及 Vc 形成了一个三级的硬件栈，用来保存 T、N、Z、C、V 标志位信息。Tc、Nc、Zc、Cc、Vc 保存当前的各标志位信息；Ts、Ns、Zs、Cs、Vs 保存前一次的各标志位信息；Tb、Nb、Zb、Cb、Vb 保存前一次的各标志位信息的备份。

异常原因寄存器（**ECR - Exception Cause Register**）：

异常原因寄存器指示了引起异常的原因。

表 1.5 异常原因寄存器（ECR – Exception Cause Register）

位	b31~b24	b23~b18	b17~b16	b15~b8	b7~b6	b5	b4~b0
读/写	R/W	R	R/W	R/W	R	R/W	R
默认值	0	0	0	0	0	0	0
名称	-	IP_H[5:0]	IP_S[1:0]	-	CE[1:0]	-	Exc_code

-	b31~b24	保留。读操作时，返回 0；写操作时，只能写 0
IP_H[5:0] ①	b23~b18	63 个硬件中断的中断请求向量
IP_S[1:0]	b17~b16	2 个软件中断的中断请求向量
-	b15~b8	保留。读操作时，返回 0；写操作时，只能写 0
CE[1:0]	b7~b6	当控制器或协处理器不可用异常发生时，这几位表明了异常发生的原因： 00：控制寄存器访问异常 01：协处理器 1 不可用异常 10：协处理器 2 不可用异常 11：协处理器 3 不可用异常
-	b5	保留。读操作时，返回 0；写操作时，只能写 0
Exc_code	b4~b0	当中断发生时，Exc_code 为异常发生原因编码

注①：IP_H[5:2]是一个 6 位编码，指示中断请求。0 表示没有中断请求，n（1~63）表明优先级为 n 的中断请求。IP[1:0]对应软件中断，用来设置或取消软件中断。

异常向量寄存器（EXCPVec - Exception Vector Register）：

异常向量寄存器保存有所有异常响量地址的基址，b0 位指示向量地址的偏移模式选择位，选择偏移 0x04 还是 0x10，可以计算出异常向量的地址。

表 1.6 异常向量寄存器(EXCPVec – Exception Vector Register)

位	b31~b16	b15~b1	b0
读/写	R/W	R/W	R/W



默认值	0x9F00	0	0
名称	EXCPVec_Base	-	VO

EXCPVec_Base　b31~b16　所有异常向量地址的基址（第 31 位至第 16 位）

-　b15~b1　保留。读操作时，返回 0；写操作时，只能写 0

VO　b0　向量地址的偏移模式选择位：

0：偏移 0x04

1：偏移 0x10

Cache 控制寄存器（CCR - Cache Control Register）：

Cache 控制器控制 Cache 的相关操作。

表 1.7 Cache 控制寄存器（CCR-Cache Control Register）

位	**b16**	**b9**	**b8**	**b7**	**b6**
读/写	R/W	R	R	R	R/W
默认值	0	0	0	0	0
名称	LDMLP	DPFB	IPFB	W-Back	RbBpDis
位	**b5**	**b4**	**b3**	**b2**	**b0**
读/写	R/W	R/W	R/W	R/W	R
默认值	0	0	0	0	0
名称	NOP	BTEN	LDM	LIM	WB

-　b31~b17　保留。读操作时，返回 0；写操作时，只能写 0

LDMLP　b16　LDM 低功耗（LDM Low power）使能位：

0：禁止（默认）

1：使能

-　b15~b10　保留。读操作时，返回 0；写操作时，只能写 0

DPFB　b9　数据预取缓存器使能位：

0：禁止

1：使能

IPFB	b8	指令预取缓存器使能位： 0：禁止 1：使能
W-Back	b7	数据 Cache 的写回（Write-Back）模式使能位： 0：禁止 1：使能
RbBpDis	b6	写缓存器的数据读操作越过写操作模式的禁止位： 0：禁止 1：使能
NOP	b5	NOP 指令的处理选择位： 0：NOP（空操作）指令正常处理 1：NOP 指令在流水线（Pipe Line）中充当“气泡”
BTEN	b4	AMBA 总线设备支持突发传输提前终止（burst early terminate）的功能使能位： 0：禁止 1：允许
LDM	b3	LDM 接口使能位： 0：禁止 Local 数据存储器（LDM，Local Data Memory）接口 1：允许 Local 数据存储器（LDM，Local Data Memory）接口
LIM	b2	LIM 接口使能位： 0：禁止 Local 指令存储器（LIM，Local Instruction Memory）接口 1：允许 Local 指令存储器（LIM，Local Instruction Memory）接口
-	b1	保留。读操作时，返回 0；写操作时，只能写 0
WB	b0	写缓冲使能位： 0：禁止（默认） 1：使能

异常程序计数器（EPC - Exception Program Counter）

异常程序计数器保存程序发生异常时的 PC 程序指针及发生异常时的指令模式。

表 1.8 异常程序计数器（EPC – Exception Program Counter）

位	b31~b1	b0
读/写	R/W	R
默认值	0	0
名称	EPC	M

EPC　　b31~b1　　异常发生时，程序计数器（PC）的高 31 位有效位被记录在此字段中。执行 RTE 指令后，此字段中的值将被重载到程序计数器 PC 中

M　　b0　　对于在流水线 D 阶段（D Stage）后产生的异常，该位表明发生异常的指令模式（32 位或 16 位）：

0：32 位指令模式

1：PCE 指令或 16 位指令模式

对于在流水线 D 阶段（D Stage）前产生的异常，该位无意义

引起中断或异常发生的指令的地址保存在 EPC 寄存器中。其中，Bit0（M）表明引起异常发生指令的模式：为 1 表示 PCE 指令或 16 位指令模式；为 0 表示 32 位指令模式。Bit1（EPC[1]）表明具体的发生异常的指令位置：为 1 表示低地址的 16 位指令；为 0 表示高地址的 16 位指令或 32 位指令。执行 RTE 指令后，EPC[31:1]的值将被重载到 PC 中，Bit0 忽略。

其他控制寄存器与用户编写程序应用关系不大，感兴趣的读者请参考 score 内核资料。

1.5 异常

1.5.1 概述

S+core 内核最多可以处理 80 个异常，以 S+core 内核构成的芯片具有强大的中断处理能力。

1.5.2 异常处理流程

S+core 微处理器在每执行一条程序前先判断是否有复位请求，如果有复位请求，会初始化程序状态寄存器（PSR）、条件寄存器（Condition）、异常向量寄存器（EXCPvec）、CCR 寄存器以及程序计数寄存器 PC；如果没有复位请求，会执行下一条指令，如果有异常请求，会将相关寄存器进行保存后转入异常程序处理；如果没有异常请求，处理器直接判断是否有复位请求等重复这个过程。

根据异常的优先级，所有异常均在流水线的 M 阶段（Memory Stage）被识别处理。在执行异常处理后，执行 RTE 指令可以使程序从异常处理中返回。执行 RTE 指令后，PC 指针将指向 EPC 寄存器中保存的地址，且部分 PSR 寄存器或条件寄存器的内容将被右移（弹出栈），CPU 即从 EPC 寄存器保存的地址处执行程序。参考图 1.3 所示：

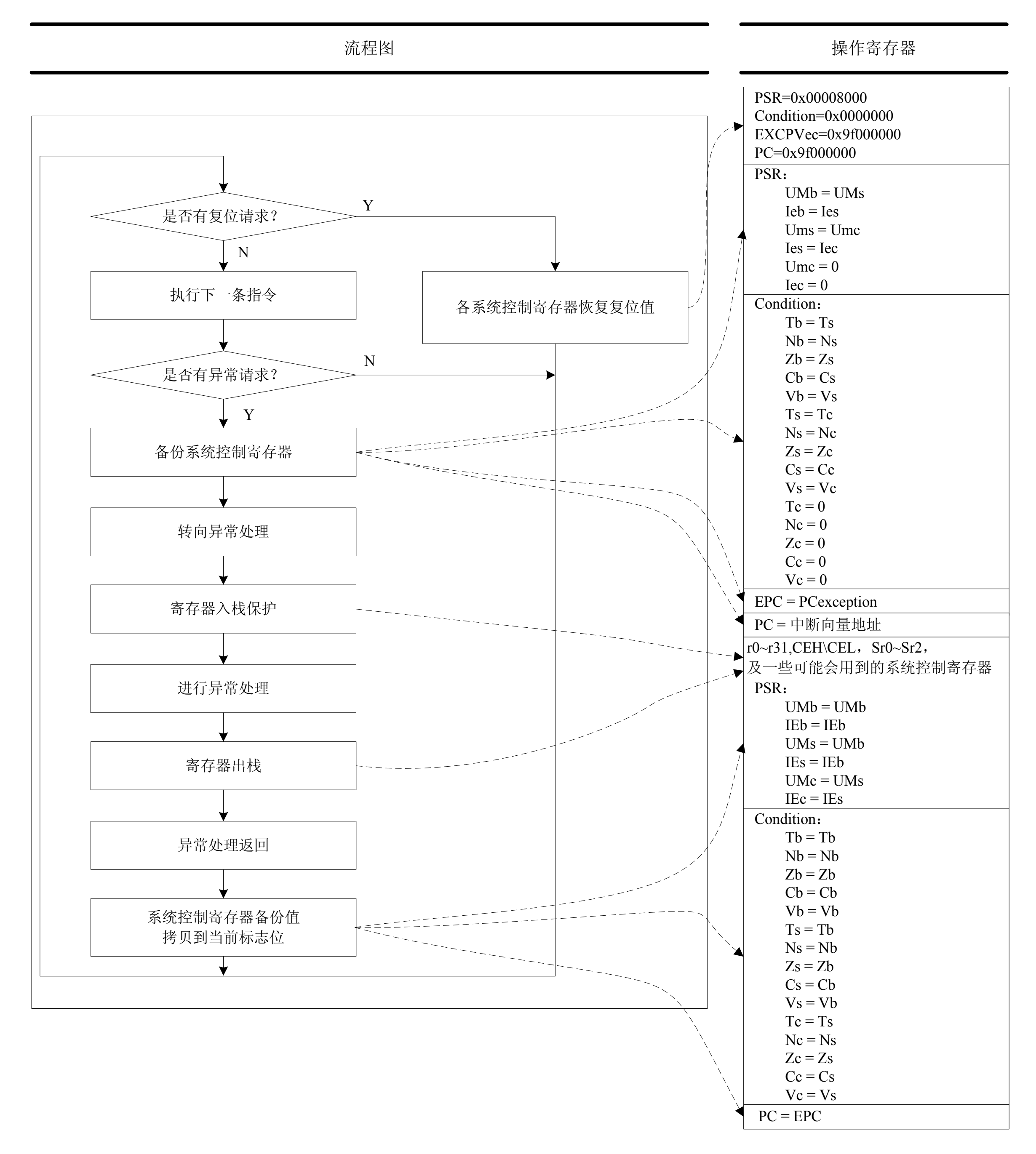


图 1.3 异常处理流程

1.5.3 异常优先级

正常模式下，在一条指令执行过程中，可能有多个异常发生，但是根据异常优先级只能有一个异常向 CPU 发出申请。

表 1.9 异常优先级

优先级	异常	描述
1 最高	Reset	复位异常



2	DSS	单步调试异常（Debug Single Step）
3	DINT	Debug 总线异常或 JTAG 中断异常
4	DDBLV	执行 Load 操作时发生的数据断点调试异常（地址数据均匹配）
5	NMI	不可屏蔽中断（Non-maskable Interrupt）
6	Interrupt[63]	外部硬件中断异常
:	:	:
:	:	:
68	Interrupt[1]	外部硬件中断异常
69	DIB	指令断点调试异常
70	AdEL-instruction	取指地址错误异常
71	BusEL-instruction	取指总线错误异常
72	P-EL	指令中的 P 位校验错误异常
73	DBP	调试断点异常（SDBBP）
73	SYSCALL	系统调用陷阱异常
73	CCU	控制器或协处理器不可用异常
73	RI	未定义指令异常
73	Trap	条件陷阱异常
74	DDBLA	执行 Load 操作时发生的数据断点调试异常
75	DDBS	执行 Store 操作时发生的数据断点调试异常
76	AdEL-Data	数据装载地址错误异常
77	AdES	数据存储地址错误异常
78	CeE	Custom Engine 执行异常（除 0 引起的）
78	CpE	协处理器执行异常
79	BusEL-data	数据访问总线错误异常
80 最低	SWI[1]	内部软件中断异常 1
80 最低	SWI[2]	内部软件中断异常 2

在 Debug 模式下，Debug 异常指令以及与中断相关的指令将被禁止。表 1.10 列出了 Debug 模

式下各种异常会引起的处理器行为。其中有些异常将导致系统再进入 Debug 模式（可以根据 DExcCode 来判断异常原因）

表 1.10 Debug 模式下异常及其优先级

优先级	异常	处理器行为
最高	复位异常	系统复位
	单步调试异常（DSS）	被禁止
	Debug 中断异常（DINT）	被禁止
	执行 Load 操作时发生的数据断点调试异常（DDBLV）地址数据均匹配	被禁止
	不可屏蔽中断异常（NMI）	被禁止
	外部硬件中断异常（Interrupt）	被禁止
	指令断点调试异常（DIB）	被禁止
	取指地址错误异常	再进入 Debug 模式
	取指总线错误异常	再进入 Debug 模式
	指令中的 P 位校验错误异常	再进入 Debug 模式
	断点调试异常（SDBBP）	被禁止
	条件陷阱异常	再进入 Debug 模式
	系统调用陷阱异常	再进入 Debug 模式
	控制器或协处理器不可用异常	再进入 Debug 模式
	未定义指令异常	再进入 Debug 模式
	执行 Load 操作时发生的数据断点调试异常（DDBLA），仅地址匹配	被禁止
	执行 Store 操作时发生的数据断点调试异常（DDBS）	被禁止
	数据装载地址错误异常	再进入 Debug 模式
	数据存储地址错误异常	再进入 Debug 模式
	Custom Engine 执行异常	再进入 Debug 模式
	协处理器执行异常	再进入 Debug 模式

	数据访问总线错误异常（精确的）	再进入 Debug 模式
	数据访问总线错误异常（不精确的）	被禁止
最低	内部软件中断异常	被禁止

1.5.4 异常原因

引起异常可以有各种原因，如硬件中断、软件中断、复位、总线异常等。引起异常的原因对应的编码，参考下表：

表 1.11 异常原因编码

异常	编码	异常	编码
Reset	0	AdEL-Data	11
NMI	1	AdES	12
AdEL-Instruction	2	-	13
-	3	-	14
-	4	-	15
BusEL-Instruction	5	CeE	16
P-EL	6	CpE	17
SYSCALL	7	BusEL-Data	18
CCU	8	SWI	19
RI	9	Interrupt	20
Trap	10		

1.5.5 异常向量

表 1.12 异常向量表

异常类型	向量地址		异常
	VO = 0	VO = 1	
复位异常	0x9F00_0000	0x9F00_0000	复位异常
Debug 异常	Badr + 0x1FC	Badr + 0x1FC	Debug 异常
一般异常	Badr + 0x200	Badr + 0x200	不可屏蔽中断异常（NMI）

			取指地址错误异常（AdEL-Instruction）
			取指总线错误异常（BusEL-Instruction）
			P 位错误异常（P-EL）
			系统调用异常（SYSCALL）
			控制器或协处理器不可用异常（CCU）
			未定义指令异常（RI）
			陷阱异常（Trap）
			数据装载地址错误异常（AdEL-Data）
			数据存储地址错误异常（AdES）
			Custom Engine 执行异常（CeE）
			协处理器执行异常（CpE）
			数据访问总线错误异常（BusEL-Data）
			内部软件中断异常（SWI1）
			内部软件中断异常（SWI2）
中断异常	Badr + 0x204	Badr + 0x210	外部硬件中断[1]
	.	.	.
	.	.	.
	Badr + 0x2FC	Badr + 0x5F0	外部硬件中断[63]

1.6 各种异常描述

1.6.1 复位异常

起因：一旦复位信号产生，总是会发生复位异常。复位异常是不可屏蔽的。

处理：CPU 为复位异常提供一个特殊的复位向量（0x9F00_0000），此向量处于非 Cache 区。进入复位异常处理后，CPU 将处于固定映射（Fixed Mapping）模式。复位异常发生后，程序状态寄存器（PSR）中的 IEc 与 UMc 被置 0，CCR 寄存器中的 WB 字段被置 0，ECR 寄存器中的 Exc_code 字段被置 0，EXCVEC 寄存器中的 EXCVec_Base 字段被置为 0x9F00_0000，其他寄存器的值均未作定义。

1.6.2 NMI 中断异常

起因：NMI 引脚电平处于下降沿时，会发生 NMI 中断（Non-Maskable Interrupt）异常，而不管

PSR 寄存器中的 IEc 位是否被置 1。NMI 中断异常是不可恢复的异常。NMI 中断异常方生后， PSR 寄存器中的 IEc 与 UMc 位的值将分别压入 IEs 与 Ums 位中；条件寄存器的 Tc、Nc、Zc、Cc、及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 1；EPC 寄存器指向引起发生 NMI 中断异常的指令，其他寄存器的值均不会改变。

1.6.3 地址错误异常

起因：装载或存储内存中的一个字时，不是字边界对齐的；装载或存储内存中的一个半字时，不是半字边界对齐的；指令地址不是半字边界对齐的（指令地址的最低位是+1）；用户模式下访问了核心模式下的地址；用户模式或核心模式下访问了 Debug 模式的地址；地址错误异常发生后，PSR 寄存器中的 IEc 与 UMc 位被压入 IEs 与 Ums；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段为 2（AdEL-Instruction 异常）、11（AdEL-Data 异常）或者 12（AdES-data 异常）；EPC 寄存器指向引起地址错误异常的指令；异常存储地址（EMA）寄存器保存没有正确对齐的或者被保护的地址空间的虚拟地址；EMA 寄存器和 PEVN 寄存器保存没有正确转换的虚拟地址，其他寄存器的值均不会改变。

1.6.4 总线错误异常

起因：当物理电路产生 Event（例如，总线访问超时、背板总线（backplane bus）P 位错误、无效的物理存储地址或者无效的访问类型信号时，会引起总线错误异常。总线错误异常发生后，PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段为 5（取指令时发生的异常）或者 18（取数据时发生的异常）；EPC 寄存器指向引起总线错误异常的指令。注意：如果处理器中有写缓存器（Write-Buffer）并已将其使能，那么，数据总线错误异常发生时，EPC 寄存器将不能正确地保存指令指针，从而该异常将不能被恢复，其它寄存器的值均不会改变。

1.6.5 陷阱异常

起因：执行 Trap 指令，会引起陷阱异常。陷阱异常发生后，PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 10；EPC 寄存器指向 Trap 指令，其他寄存器的值均不会改变。

1.6.6 系统调用（SYSCALL）异常

起因：执行 SYSCALL 指令，会引起系统调用陷阱异常。系统调用异常发生后， PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 7；EPC 寄存器指向 SYSCALL 指令，其它寄存器的值均不会改变。

1.6.7 P-EL 异常

起因：执行一条 P 位校验错误指令，会引起 P-EL 异常；指令地址（如分支目标地址）指向一个 32 位指令字的低半字，但其是一条 32 位指令，即指令地址字对齐错误（Bit1 = 1）；P-EL 异常发

生后，PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 6；EMA 寄存器指向引起 P-EL 中断异常的指令；EPC 寄存器指向引起 P-EL 中断异常的指令，其它寄存器的值均不会改变。

1.6.8 未定义指令（RI）异常

起因：执行一条操作码（Opcode）未被定义的指令，会引起 RI 异常。RI 异常发生后， PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 9；EPC 寄存器指向引起该异常的指令，其它寄存器的值均不会改变。

1.6.9 控制器或协处理器不可用（CCU）异常

起因：执行如下指令操作中的任一种时，会引起 CCU 异常。在用户模式下，执行控制寄存器指令，但控制寄存器没有标志为可用状态（PSR 寄存器 CRA 位为 0）；执行协处理器指令，但该协处理器没有标识为可用状态（PSR 寄存器 CU 位为 0）。CCU 异常发生后， PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 8；ECR 寄存器中的 CE 字段表明具体是哪个协处理器或控制器不可用；EPC 寄存器指向引起 CCU 异常的指令，其它寄存器的值均不会改变。

1.6.10 Custom Engine 执行（CeE）异常（除 0 引起）

起因：Custom Engine 执行一条扩展指令时，执行结果可能会引起 CeE 异常。例如，除 0、乘累加溢出。这里，定义除 0 引起的异常为默认的 CeE 异常。CeE 异常发生后，PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 16；EPC 寄存器指向引起 CeE 异常的指令，其它寄存器的值均不会改变。

1.6.11 协处理器 z 执行（CpE）异常

起因：协处理器 z 试图执行一条协处理器操作指令时，执行结果可能会引起 CpE 异常。例如，浮点除 0、浮点乘累加溢出。CpE 异常发生后，PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段被置 17；ECR 寄存器中的 CE 字段表明具体是哪个协处理器引起的异常；EPC 寄存器指向引起 CeE 异常的指令，其它寄存器的值均不会改变。

1.6.12 中断异常

起因：当 65 个中断中的任一个发生时，会引起中断异常。中断异常发生后，PSR 寄存器中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；条件寄存器的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位；ECR 寄存器中的 Exc_code 字段为 19（表明是软件中断引起的异常）或 20（表明是硬件中断引起的异常）；EPC 寄存器指向引起中断异常的指令。运行完中断服务程序

后，程序会返回到 EPC 寄存器所指的地址，其它寄存器的值均不会改变。

注意：在分配外部中断（63 个）优先级之前，中断控制器必须先利用处理器时钟去捕获外部中断信号并将其杂散信号去抖，否则这些杂散信号可能会引发处理器错误地识别中断。

1.6.13 Debug 中断异常

起因：非 Debug 模式下，Debug 中断会引发该异常。Debug 中断异常发生后，DREG 寄存器（CR29）中的 DINT 以及 DM 位被置位；DEPC 寄存器（CR30）保存异常处理完成后需要恢复的地址。执行 DRTE 指令，可使程序跳转到该寄存器中所指的地址，其它寄存器的值均不会改变。

1.6.14 单步调试异常

起因：非 Debug 模式下，当一条指令单步执行完并且 Debug 寄存器中的 SSEn 位被置位后，会发生单步调试异常。单步调试异常发生后，DREG 寄存器（CR29）中的 DSS 以及 DM 位被置位；DEPC 寄存器（CR30）保存异常处理完成后需要恢复的地址。执行 DRTE 指令，可使程序跳转到该寄存器中所指的地址，其它寄存器的值均不会改变。

1.6.15 断点调试异常

起因：执行 SDBBP（Software Debug Breakpoint）指令后，会发生断点调试异常。断点调试异常发生后，如果该异常发生在非 Debug 模式下，DREG 寄存器（CR29）中的 DBP 位被置位；如果该异常发生在 Debug 模式下，DSS、DBP、DDBL、DDBS、DIB、DDB 和 DINT 这些位被清除，在 CR29 寄存器中 DexcCode 被设置成 SDBBP；DREG 寄存器（CR29）中的 DM 位被置位；DEPC 寄存器（CR30）保存异常处理完成后需要恢复的地址。执行 DRTE 指令，可使程序跳转到该寄存器中所指的地址，其它寄存器的值均不会改变。

1.6.16 数据断点调试异常

起因：非 Debug 模式下，执行装载或存储指令时，若被存取的数据的地址匹配，则会发生数据断点调试异常。据断点调试异常发生后，DREG 寄存器（CR29）中的 DDBLA、DDBLV 或 DDBS 位被置位；DREG 寄存器（CR29）中的 DM 位被置位，其它寄存器的值均不会改变。

DEPC 寄存器（CR30）保存异常处理完成后需要恢复的地址。执行 DRTE 指令，可使程序跳转到该寄存器中所指的地址。

1.6.17 指令断点调试异常

起因：非 Debug 模式下，若指令的地址匹配，则会发生指令断点调试异常。指令断点异常发生后，DREG 寄存器（CR29）中的 DIB 以及 DM 位被置位；DEPC 寄存器（CR30）保存异常处理完成后需要恢复的地址。执行 DRTE 指令，可使程序跳转到该寄存器中所指的地址，其它寄存器的值均不会改变。

1.7 缓存简介

S+core7 处理器的主频最高达到 162MHz，而主存储器操作用动态存储器（DRAM），其存储周期仅为 100ns~200ns。这样，如果指令和数据都存储在主存储器中，主存储器的速度将会严重制约

整个系统的性能。高速缓冲存储器（Cache）和写缓冲区（Write buffers）位于主存储器和 CPU 之间，可以大大提高存储系统的性能。Cache 是一块地址可以改变的高速的存储器空间，目的在于加快存储器访问的速度。将 Cache 分成若干块，Cache-line 是使用 Cache 的最小存储单元。当 CPU 读取数据或者指令时，如果当前 cache line 中没有保存这些数据，cache line 会将这些数据载入。如果在 cache line 载入其它地址的数据之前，CPU 访问相同的数据，那么 cache 可以提供存储器访问。

S+core 处理器支持 2 种独立的 Cache，指令 Cache（I-Cache）和数据 Cache（D-Cache）。这种独立的 Cache 结构可使指令和数据同时得到处理。而 I-Cache 和 D-Cache 均是使用虚拟地址来索引的（virtual indexed），用物理地址来标识的（physically tagged）。

表 1.13 S+core 处理器 Cache 规格

规格	I-Cache	D-Cache
Cache 大小	4Kbytes	4Kbytes
组相连	2	2
Cache-line 大小	4Words	4Words
写策略	NA	写透（Write Through）或写回（Write Back）
分配策略	读分配策略（Read Allocate）	读分配策略（Read Allocate）
替换策略	最近最少使用策略（LRU）	最近最少使用策略（LRU）
预取一个 Cache-line	虚拟地址模式	虚拟地址模式
预取并锁定一个 Cache-line	虚拟地址模式	虚拟地址模式
使一个 Cache-line 无效	虚拟地址模式	虚拟地址模式
使整个 Cache 无效	可以	可以
清空写缓冲区	NA	可以
写缓冲区	NA	4Words

S+core 处理器中，Tag 存储陈列是以 Cache-line 为单位标记数据的；数据是以 Word 为单位存储的。I-Cache 和 D-Cache 均是 2 路组相连的结构，大小均为 4K Bytes。

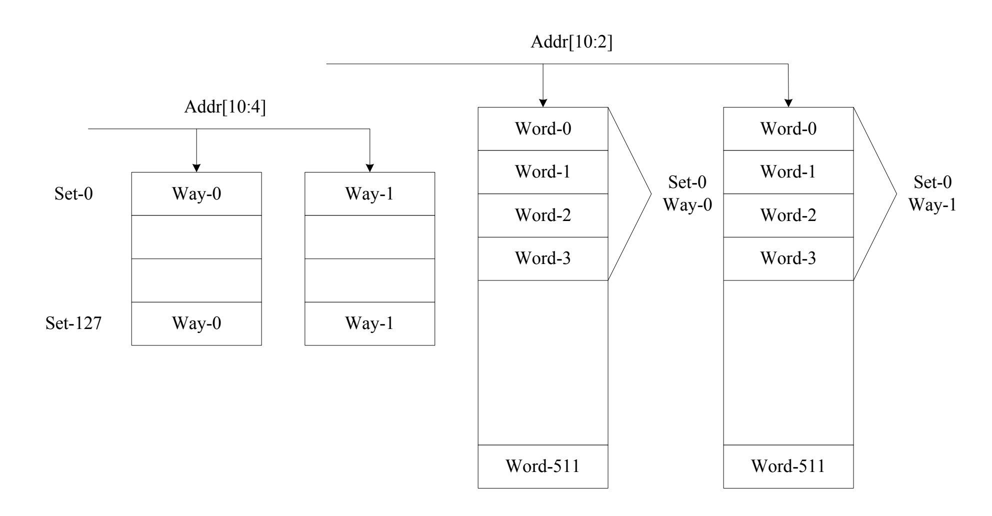


图 1.4 S+core Cache 存储阵列

1.7.1 指令 Cache

I-Cache 控制器是一个独立于处理器内核之外的设备。在内核与 I-Cache 控制器之间有两条请求总线：一条是指令总线（I-Bus），另一条是 I-Cache 指令总线。I-Cache 控制器需要处理来自这两条总线的请求。若它不能在下一个 Cycle 处理完请求，I-Cache 控制器将阻塞流水线（pipeline）去继续处理请求。

I-Cache 特性：

- 2 路组相联的 Cache 结构
- Cache 的大小是 4K bytes
- Cache-line 的大小是 4Words
- 采用读操作分配策略
- 提供使一个 Cache-line 无效的命令（虚拟地址模式下）
- 提供使整个 Cache 内容无效的命令（虚拟地址模式下）
- 提供预取一个 Cache-line 的命令（虚拟地址模式下）
- 提供预取和“Lock”一个 Cache-line 的命令（虚拟地址模式下）
- 在内核和 I-Cache 控制器之间有指令预取请求总线和 I-Cache 请求总线
- 取指操作时提供精确总线错误异常（precise bus error exception）
- 处理器内核可以拒绝 I-Cache 设备的当前和前一指令请求

1.7.2 数据 Cache

D-Cache 控制器也是一个独立于处理器内核之外的设备。在内核与 D-Cache 控制器之间也有两条请求总线（Request Bus）：一条是用于 Load 或 Store 指令的总线（D-Bus），另一条是 D-Cache 指

令总线。D-Cache 控制器需要处理来自这两条总线的请求。若 D-Cache 控制器不能在下一个 Cycle 处理完请求，它将阻塞流水线（pipeline）。

D-Cache 特性：

- 2 路组相联的 Cache 结构
- 4K bytes 的 Cache 大小
- 4 Words 的 Cache line 大小
- 读操作分配策略
- 提供使 Cache line 无效的命令
- 提供预取 Cache line 的命令
- 提供预取并锁定 Cache line 的命令
- Load 操作时提供精确的总线异常；若写缓冲器是禁用的，Store 操作时也提供精确的总线异常
- 若写缓冲器是允许的，Store 操作时则提供不精确的总线异常

1.7.3 存储器一致

系统设计时必须考虑存储一致性问题。因为 Cache 中保存的是主存数据的一块拷贝，而当 Cache 中数据正被使用时，其它存储设备可能去修改主存中的那块数据，从而使 Cache 中的数据与主存中的不一致。存储器一致性问题必须通过系统设计或软件来解决。下面是 4 种典型的存储一致性问题：

写指令操作：当处理器内核存储指令到主存中以待后续执行时，必须保证这些指令被写到主存中，并且保证 I-Cache 中的相应位置上的内容是无效的。S+core 处理器中，D-Cache 与 I-Cache 是独立的。

写回 Cache 操作：当一个 DMA 设备直接从主存中移出数据时，能获得正确的数据是至关重要的。如果 D-Cache 使用的是写回策略，且程序最近曾执行写数据操作，那么，要移出的正确数据可能还保存在 D-Cache 中，未被写入主存。而内核是不清楚这种情况的。因此，必要的话，在 DMA 设备开始从主存中读数据之前，必须将还保存在 D-Cache 中的正确数据写入主存中。

从 DMA 设备装载数据到主存：当从一个 DMA 设备中装载数据到主存中时，使 Cache 中在地址范围内的所有 Cache line 无效是至关重要的。否则，当内核从 DMA 设备中读数据时，会读到保存在 Cache 中的旧的数据。因此，在内核从 DMA 设备中读数据之前，必须使相应得 Cache line 无效。

写缓冲器：当写缓冲器被允许且其中已有一些数据时，主存中内容或 IO 寄存器值可能是不正确的。为达到存储数据的一致性，应该通过 Cache 指令将写缓冲器清空。

1.8 LIM 和 LDM

1.8.1 LIM-Local Instruction Memory

LIM（指令存储器）系统包括位于 I-Cache 控制器中的 SRAM（Synchronous RAM）。SRAM 提供了访问存储块的快速接口。用户可以通过 CCR 控制寄存器的 Bit2 来使能 LIM。注意，LIM 设备

被允许并使用之前，必须正确初始化。S+core 处理器提供了可以初始化 SRAM 或向 SRAM 填充内容的 Cache 指令。

1.8.2 LDM-Local Data Memory

LDM（数据存储器）系统也包括 SRAM（Synchronous RAM）。SRAM 提供了访问存储块的快速接口。用户可以通过 CCR 控制寄存器的 Bit3 来使能 LDM。注意，LDM 设备被允许并使用之前，必须正确初始化。S+core 处理器提供了可以初始化 SRAM 或向 SRAM 填充内容的 Cache 指令。

LDM 的地址范围可以通过 Cache 指令（填充 LDM）来配置，当执行“填充 LDM”指令时，D-Cache 控制器可以记录 LDM 设备的物理地址。

1.9 片上调试

S+core 处理器的 Debug 机制具备如下特征：

片外 Debug Memory 访问

SJTAG 允许处理器在 Debug 模式下访问片外的指令或数据。处理器发出的访问申请经 JTAG 协议解析后发送到由 Debug Host PC 控制的 Debug Probe 中。然后，Debug Probe 再将要访问的地址重定位到 Probe Memory 的地址。接下来，对应地址中的数据就会再经过 JTAG 然后被读到处理器中。

通过这种机制，不需要芯片具备 Debug ROM 就可以进行系统调试，从而实现了处理器与 Debug Host PC 间的通信。

硬件断点

提供两种硬件断点。用户可配置这两种硬件断点，以便在下列情况时发生 Debug 异常：

从特定地址中取指令（Breakpoint）

从特定地址中取数据与访问数据（Watchpoint）

软件断点指令

提供两个辅助的指令：软件断点调试（SDBBP，Software Debug Breakpoint）和从 Debug 异常返回（DRET，Debug Exception Return）。这两个指令可辅助系统调试。

单步执行

通过处理器提供的单步异常机制，可实现一条指令一条指令地执行程序，从而进行程序调试。

Debug 中断

Debug 中断用于强制处理器在任何时候进入 Debug 模式

DMA 访问

由 SJTAG 直接控制 DMA 通道是透过 BIU 来访问系统总线的。当用户需要直接下载代码到系统存储器中时，这一特性很有用。

2 S+core7 指令系统

2.1 概述

指令是 CPU 执行某种操作的命令。微处理器（MPU）或微控制器（MCU）所能识别全部指令的集合成为指令系统，是研制厂家在设计 CPU 时所赋予的功能。只有正确书写和使用指令，才能完成设计任务。因此，学习和掌握指令的功能与应用是程序设计的基础。本章将详细介绍 S+core 的指令系统。

S+core 指令系统分为 32 位指令集和 16 位指令集。

32 位指令集的效率高，功能强，可以完成所有处理器支持的操作。大多数 32 位指令都具有三个操作数，且支持使用 16 位的立即数。另外，32 位指令可以访问全部 32 个通用寄存器。32 位指令集包含一些系统控制指令以及协处理器控制指令，这些指令是 16 位指令集不具备的。这些指令可以控制处理器完成一些复杂处理，大大增强系统的功能。

16 位指令集可以看作是 32 位指令集的压缩形式的子集，它是专为编译器（compiler）编译混合模式的指令而设计，以减少代码容量的，一般不推荐用来进行手动汇编编码。由于 16 位指令集的指令宽度的限制，基本上所有的 16 位指令都是两个操作数的操作，且允许使用小立即数，另外，由于指令长度的限制，指令只能访问 32 个通用寄存器中的前 16 个（r0～r15）。当发生 16 位指令不为字对齐的情况，则最好将该 16 位指令改为相应的 32 位指令，而最好不要插入一个 16 位的 nop!指令来对齐字边界，因为这样将增加指令执行个数。

32 位指令集包括装载与存储指令、数据处理指令、跳转与分支指令、特殊指令和协处理器指令五类指令。其中装载与存储指令可以完成对存储器的存取操作；数据处理指令完成算术和逻辑运算等操作；分支指令完成程序跳转以及函数调用等操作；特殊指令包含了对 Cache 的操作以及对处理器内核的特殊寄存器的控制等；协处理器指令可以完成一些图像编码等特殊功能。

16 位指令集包括装载与存储指令、数据处理指令、跳转与分支指令和特殊指令四类指令。

2.2 指令格式与编码

S+core 处理器是基于精简指令集计算机（RISC）原理设计的，指令集和相关译码机制较为简单。S+core 处理器具有 32 位的指令总线，每次将 32 位长度的指令代码填入译码机构。 S+core 处理器支持 32 位和 16 位两种指令模式。处理器可以自动识别一个 32 位的编码是一条 32 位的指令还是两条 16 位的指令，而不需要任何指令模式的切换。一个 32 位的指令编码中的 bit31 和 bit15 两位称为“P-位”，用来区分指令模式。所以，32 位指令的实际有效宽度为 30 位，而 16 位指令的实际有效宽度为 15 位。如图 2.1 所示。

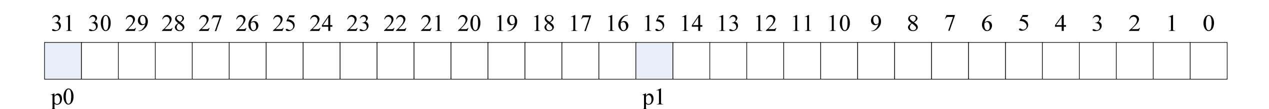


图 2.1 S+core 处理器的指令模式

对应于 P-位的不同编码，指令格式的意义如表 2.1 所示。

表 2.1 P-位指示的指令格式

p0	p1	意义	格式符号
1	1	32 位指令	32BF
0	0	16 位指令＋16 位指令	16BF
0	1	并行条件执行（PCE-Parallel Condition Execute）指令 并行执行条件标志 T 为 1 时执行高半字的 16 位指令 1 并行执行条件标志 T 为 0 时执行低半字的 16 为指令 2	PCEF
1	0	未定义	UDEF

除去 P-位之后的指令被划分为若干字段，每一字段均代表一定的含义。其中，32 位指令的主操作码由第 25～29 位确定，16 位指令的主操作码则由第 12～14 位确定。指令中其余字段的意义随指令的不同而有所不同。各类指令包含的字段如图 2.2 所示。

30 bit	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
J-form	OP					Disp24(lmm)																								LK
BC-form	OP					Disp[18:9](lmm)										BC					Disp[8:0](lmm)									LK
Special-form	OP					rD					rA					rB					0	0	0	func6						CU
I-form	OP					rD(D,S1)					func3			Imm16(S2)																CU
RI-form	OP					rD(D)					rA(S1)					Imm14(S2)														CU
RIX-form	OP					rD(D)					rA(S1)					Imm12(S2)												func3		
CENew	OP					USD1					rA(optional)					rB(optional)					USD2					func5				
CR-form	OP					rD					CR					0	0	0	0	0	0	0	CR_OP			func2		II	0	MI

coprocessor	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mtc/mfc	OP					rD					CrA					0	0	0	0	0	0	0	0	0	0	CP#		Sub-OP		
ldc/stc	OP					rD					CrA					Imm10										CP#		Sub-OP		
cop	OP					CrD					CrA					CrB					COP-Code					CP#		Sub-OP		

15 bit	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B<cond>x	OP			EC				Disp8(Imm)							
J-form	OP			Disp11(Imm)											LK
R-form	OP			rD_{go}				rA_{go}				func4			
I-form-1	OP			rD_{go}				Imm5					func3		
I-form-2	OP			rD_{go}				Imm8							



图 2.2 S+core 处理器指令字段

S+core 指令集按照 OP 字段对指令进行分类，32 位指令集可以分为十一类，分别是：J-form、BC-form、Special-form、I-form、RI-form、RIX-form、CENew、CR-form、mtc/mfc 类、ldc/stc 类以及 cop 类等；16 位指令集可以分为五类，分别是：B<cond>x、J-form、R-form、I-form-1、I-form-2 等。表 2.2 所示的是各分类的具体解释。如，J-form 表示无条件跳转类的指令。

表 2.2 指令分类对照表

符号	含义
J-form	无条件跳转类指令
BC-form	条件跳转类指令（32 位指令）
Special-form	寄存器做为源操作数的指令（32 位指令）
I-form	立即数做为源操作数的指令
RI-form	寄存器与立即数做为源操作数的指令
RIX-form	变址寻址类指令
CENew	用户扩展类指令
CR-form	特殊功能寄存器操作类指令
B<cond>x	条件跳转类指令（16 位指令）
R-form	寄存器做为源操作数的指令（16 位指令）

为了描述方便，对在指令系统叙述过程中所要用到的符号作统一约定，如表 2.3 所示。

在 S+core 的指令系统中，根据指令的功能不同，指令具有的语法形式也是不同的。一条典型的 32 位指令语法格式如下所示：

<opcode>{.c} {operand1}, {operand2}

在上面的格式中，“<>”符号内的项是必须的，“{}”符号内的项是可选的。例如，<opcode>是指令助记符，这里必须书写；而{.c}是可选开关，指该指令执行结束后是否需要对条件标志位产生影响(针对某些指令，可以选择该项)；operand1 一般是一个通用寄存器，用于保存运算结果；operand2 可以是一个或者两个通用寄存器，或者是寄存器与立即数的组合，或者是一个立即数，用于参与运算，或者指定寻址地址。

例：

and.c r4, r2, r1

其中，and 是必须的，表示该指令完成的操作；.c 是可选的，表示该操作将影响条件标志位，这里，operand1 是 r4，用来保存与操作之后的结果；operand2 是 r2 和 r1，表示 r2 与 r1 做与操作。

表 2.3 指令字段含义说明

字段	含义描述
OP	主操作码
func	扩展功能码

CU	条件标志更新位 0：不更新条件标志 1：更新条件标志，以反应操作的结果
LK	链接位 0：不更新链接寄存器（r3）； 1：更新链接寄存器（r3)。若指令为分支指令，则分支指令的下一条指令的地址将被置于链接寄存器（r3）内。
BN	位操作指令的指定操作位数
BC	分支指令的分支条件（Branch Condition）
EC	条件执行指令的执行条件码（Execute Condition）
TC	T 标志更新指令的测试条件码
SA	移位/循环指令的移位位数
rD	目标通用寄存器（GPR）
rA，rB	源通用寄存器（GPR）
g0	代表 GPR0~15
g1	代表 GPR16~31
Disp	跳转和分支指令的偏移量
Imm	立即数
CR	控制寄存器
Sub-OP	协处理器指令的扩展子操作码
USD	扩展定制（custom）指令的用户定义字段
CrA	源协处理器寄存器
CrD	目标协处理器寄存器
CP#	协处理器编号
COP-code	协处理器指令的扩展操作码
Exp	指数
Spar（Imm）	用于陷阱及其系统调用指令传递信息的程序参数
Srn	特殊寄存器编号

2.3 32 位指令集

32 位指令集包括装载与存储指令、数据处理指令、分支指令、特殊指令和协处理器指令五类指令。其中装载与存储指令可以完成对存储器的存取操作；数据处理指令完成算术和逻辑运算等操作；分支指令完成程序跳转以及函数调用等操作；特殊指令包含了对 Cache 的操作以及对处理器内核的特殊寄存器的控制等；协处理器指令可以完成一些图像编码等特殊功能。下面将分别对这五类指令做介绍。

2.3.1 装载与存储指令

S+core 处理器是装载/存储体系结构的典型的 RISC 处理器，对存储器的访问只能通过使用装载和存储指令实现。S+core 的装载/存储指令可以实现字、半字、无符号/有符号字节操作。

装载指令用于从内存中读取数据放入寄存器中；存储指令用于将寄存器中的数据保存到内存。S+core 有两类装载/存储指令，一类用于对齐数据地址存储单元的存取操作，另一类用于非对齐数据地址存储单元的存取操作。表 2.4 所列为 S+core 的装载/存储指令表。

表 2.4 S+core 装载/存储指令表

助记符	说明	格式
lb	装载字节数据	lb rD, [rA, SImm15]
	装载字节数据(后变址寻址)	lb rD, [rA]+, SImm12
	装载字节数据(前变址寻址)	lb rD, [rA, SImm12]+
lbu	装载无符号字节数据	lbu rD, [rA, SImm15]
	装载无符号字节数据(后变址寻址)	lbu rD, [rA]+, SImm12
	装载无符号字节数据(前变址寻址)	lbu rD, [rA, SImm12]+
lh	装载半字数据	lh rD, [rA, SImm15]
	装载半字数据(后变址寻址)	lh rD, [rA]+, SImm12
	装载半字数据(前变址寻址)	lh rD, [rA, SImm12]+
lhu	装载无符号半字数据	lhu rD, [rA, SImm15]
	装载无符号半字数据(后变址寻址)	lhu rD, [rA]+, SImm12
	装载无符号半字数据(前变址寻址)	lhu rD, [rA, SImm12]+
lw	装载字数据	lw rD, [rA, SImm15]
	装载字数据(后变址寻址)	lw rD, [rA]+, SImm12
	装载字数据(前变址寻址)	lw rD, [rA, SImm12]+

助记符	说明	格式
sb	存储字节数据	sb rD, [rA, SImm15]
	存储字节数据(后变址寻址)	sb rD, [rA]+, SImm12
	存储字节数据(前变址寻址)	sb rD, [rA, SImm12]+
sh	存储半字数据	sh rD, [rA, SImm15]
	存储半字数据(后变址寻址)	sh rD, [rA]+, SImm12
	存储半字数据(前变址寻址)	sh rD, [rA, SImm12]+
sw	存储字数据	sw rD, [rA, SImm15]
	存储字数据(后变址寻址)	sw rD, [rA]+, SImm12
	存储字数据(前变址寻址)	sw rD, [rA, SImm12]+
ldi	装载立即数	ldi rD, SImm16
ldis	装载移位立即数	ldis rD, SImm16
lcb	装载合并字数据开始	lcb [rA]+
lcw	装载合并字数据	lcw rD, [rA]+
lce	装载合并字数据结束	lce rD, [rA]+
scb	存储合并字数据开始	scb rD, [rA]+
scw	存储合并字数据	scw rD, [rA]+
sce	存储合并字数据结束	sce [rA]+

装载/存储指令有以下几种寻址方式：立即数寻址、基址变址寻址、前变址寻址和后变址寻址。

[1] 立即数寻址的格式如下：

<opcode> <rD>, <SImm>

立即数寻址是将立即数作为操作数直接参与操作。

[2] 基址变址寻址方式的格式如下：

<opcode> <rD>, <[rA, SImm15]>

基址变址寻址是将源寄存器 rA 的内容与后面给出的立即数偏移量相加而形成操作数的有效地址。

[3] 前变址寻址格式如下：

<opcode> <rD>, <[rA, SImm12]+>

前变址寻址是将源寄存器 rA 的内容与后面给出的立即数偏移量相加而形成操作数的有效地址，

并在操作完毕后将 rA 的内容更新为本次操作的有效地址。

[4]　后变址寻址格式如下：

<opcode> <rD>, <[rA]+, SImm12>

后变址寻址是指将源寄存器 rA 的内容作为操作数的有效地址，并在操作完毕后，rA 的内容通过与后面给出的立即数相加而更新。

lb/sb——装载/存储字节指令

使用 lb 指令可以将一个有符号的字节数据从存储器装载到寄存器，sb 可以将一个字节数据从寄存器保存到存储器。lb 和 sb 指令都具有基址变址寻址、前变址寻址和后变址三种寻址方式的指令格式。指令格式如下：

[1]　lb rD, [rA, SImm15]

基址变址寻址。从存储器[rA+SImm15]地址处读取字节数据并做符号扩展，然后保存至 rD 寄存器。例如：lb r4, [r2, 0x1234]。

[2]　lb rD, [rA]+, SImm12

后变址寻址。从存储器[rA]地址处读取字节数据并做符号扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容与 SImm12 相加并更新。例如：lb r4, [r2]+, 0x4。

[3]　lb rD, [rA, SImm12]+

前变址寻址。从存储器[rA+SImm12]地址处读取字节数据并做符号扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：lb r4, [r2, 0x0123]+。

[4]　sb rD, [rA, SImm15]

基址变址寻址。将 rD 寄存器低 8 位保存至存储器[rA+SImm15]地址处。例如：sb r4, [r2, 0x123]。

[5]　sb rD, [rA]+, SImm12

后变址寻址。将 rD 寄存器低 8 位保存至存储器[rA]地址处。操作完成后 rA 的内容与 SImm12 相加并更新。例如：sb r4, [r2]+, 0x4。

[6]　sb rD, [rA, SImm12]+

前变址寻址。将 rD 寄存器低 8 位保存至存储器[rA+SImm12]地址处。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：sb r4, [r2, 0x0123]+。

lbu——装载无符号字节指令

使用 lbu 指令可以将一个无符号的字节数据从存储器装载到寄存器。指令格式如下：

[1]　lbu rD, [rA, SImm15]

基址变址寻址。从存储器[rA+SImm15]地址处读取字节数据并做零扩展，然后保存至 rD 寄存器。例如：lbu r4, [r2, 0x1234]。

[2]　lbu rD, [rA]+, SImm12

后变址寻址。从存储器[rA]地址处读取字节数据并做零扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容与 SImm12 相加并更新。例如：lbu r4, [r2]+, 0x4。

[3] lbu rD, [rA, SImm12]+

前变址寻址。从存储器[rA+SImm12]地址处读取字节数据并做零扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：lbu r4, [r2, 0x0123]+。

lh/sh——装载/存储半字指令

使用 lh 指令可以将一个有符号的半字数据从存储器装载到寄存器，sh 可以将一个半字数据从寄存器保存到存储器。需要注意的是，存储器地址必须为半字对齐，否则将发生地址对齐错误异常。lh 和 sh 指令都具有基址寻址、前变址寻址和后变址三种寻址方式的指令格式。指令格式如下：

[1] lh rD, [rA, SImm15]

基址变址寻址。从存储器[rA+SImm15]地址处读取半字数据并做符号扩展，然后保存至 rD 寄存器。例如：lh r4, [r2, 0x1234]。

[2] lh rD, [rA]+, SImm12

后变址寻址。从存储器[rA]地址处读取半字数据并做符号扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容与 SImm12 相加并更新。例如：lh r4, [r2]+, 0x4。

[3] lh rD, [rA, SImm12]+

前变址寻址。从存储器[rA+SImm12]地址处读取半字数据并做符号扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：lh r4, [r2, 0x0124]+。

[4] sh rD, [rA, SImm15]

基址变址寻址。将 rD 寄存器低 16 位保存至存储器[rA+SImm15]地址处。例如：sh r4, [r2, 0x124]。

[5] sh rD, [rA]+, SImm12

后变址寻址。将 rD 寄存器低 16 位保存至存储器[rA]地址处。操作完成后 rA 的内容与 SImm12 相加并更新。例如：sh r4, [r2]+, 0x4。

[6] sh rD, [rA, SImm12]+

前变址寻址。将 rD 寄存器低 16 位保存至存储器[rA+SImm12]地址处。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：sh r4, [r2, 0x0124]+。

lhu——装载无符号半字指令

使用 lhu 指令可以将一个无符号的半字数据从存储器装载到寄存器。指令格式如下：

[4] lhu rD, [rA, SImm15]

基址变址寻址。从存储器[rA+SImm15]地址处读取半字数据并做零扩展，然后保存至 rD 寄存器。例如：lhu r4, [r2, 0x1234]。

[5] lhu rD, [rA]+, SImm12

后变址寻址。从存储器[rA]地址处读取半字数据并做零扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容与 SImm12 相加并更新。例如：lhu r4, [r2]+, 0x4。

[6] lhu rD, [rA, SImm12]+

前变址寻址。从存储器[rA+SImm12]地址处读取半字数据并做零扩展，然后保存至 rD 寄存器。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：lhu r4, [r2, 0x0123]+。

lw/sw——装载/存储字指令

使用 lw 指令可以将一个字数据从存储器装载到寄存器，sw 可以将一个字数据从寄存器保存到存储器。需要注意的是，存储器地址必须为字对齐，否则将发生地址对齐错误异常。lw 和 sw 指令都具有基址寻址、前变址寻址和后变址三种寻址方式的指令格式。指令格式如下：

[1] lw rD, [rA, SImm15]

基址变址寻址。从存储器[rA+SImm15]地址处读取字数据，并保存至 rD 寄存器。例如：lw r4, [r2, 0x1234]。

[2] lw rD, [rA]+, SImm12

后变址寻址。从存储器[rA]地址处读取字数据，并保存至 rD 寄存器。操作完成后 rA 的内容与 SImm12 相加并更新。例如：lw r4, [r2]+, 0x4。

[3] lw rD, [rA, SImm12]+

前变址寻址。从存储器[rA+SImm12]地址处读取字数据，并保存至 rD 寄存器。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：lw r4, [r2, 0x0124]+。

[4] sw rD, [rA, SImm15]

基址寻址。将 rD 寄存器内容保存至存储器[rA+SImm15]地址处。例如：sw r4, [r2, 0x124]。

[5] sw rD, [rA]+, SImm12

后变址寻址。将 rD 寄存器内容保存至存储器[rA]地址处。操作完成后 rA 的内容与 SImm12 相加并更新。例如：sw r4, [r2]+, 0x4。

[6] sw rD, [rA, SImm12]+

前变址寻址。将 rD 寄存器内容保存至存储器[rA+SImm12]地址处。操作完成后 rA 的内容更新为本次操作的存储器地址。例如：sw r4, [r2, 0x0124]+。

ldi——装载 16 位立即数

指令格式：lbi rD, SImm16

使用 ldi 指令可以将一个 16 位的立即数进行符号扩展并装载至目标寄存器 rD。ldi 指令只具有立即数寻址一种寻址方式。

例如，使用 ldi 指令进行立即数加法运算的程序如程序清单 2.1 所示。

程序清单 2.1 立即数加法

```
.data
sum: .word 0x0          // 定义变量
.text
...
la r4, sum              // 取得变量 sum 的地址
ldi r2, 0x1234          // 立即数装入 r2
ldi r3, 0x5678          // 立即数装入 r3
add r1, r2, r3          // 加法
sw r1, [r4, 0]          // 结果保存至 sum 变量
...
```

ldis——装载移位立即数

指令格式：lbis rD, Imm16

使用 ldsi 指令可以将一个 16 位的立即数左移 16 位后装载至目标寄存器 rD。ldis 指令只具有立即数寻址一种寻址方式。

例如：ldi r4, 0xabcd

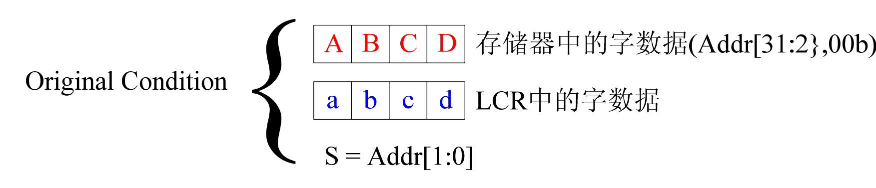
lcb——装载合并字数据开始

指令格式：lcb [rA]+

lcb 指令将存储器中指定地址的数据装载入特殊寄存器 LCR（Load Combine Register）中，操作地址由 rA&0xfffffffc 确定。完成操作后，寄存器 rA 的内容自动增加 4。由于在对存储器进行存取操作时 rA 的最低两位被截去，所以不会发生地址对齐错误。该指令与 lcw、lce 配合可以完成对非对齐地址数据的存取。

图 2.3 说明了当特殊寄存器 LCR 和存储器中各给定一字数据时，装载合并指令的操作。

注意：图 2.3 图 2.4 仅针对 SPCE3200 如此，以后 Score 内核可能有所更改，请读者注意。



	Big Endian (s = 0 1 2 3)				Little Endian (s = 3 2 1 0)			
	Mode (Big)	rD	Next LCR	Load Word?	Mode (Little)	rD	Next LCR	Load Word?
LCB (LCB [Addr])	s=0	(Not Set)	ABCD	YES	s=0	(Not Set)	ABCD	YES
	s=1	(Not Set)	ABCD	YES	s=1	(Not Set)	ABCD	YES
	s=2	(Not Set)	ABCD	YES	s=2	(Not Set)	ABCD	YES
	s=3	(Not Set)	ABCD	YES	s=3	(Not Set)	ABCD	YES
LCW (LCW rD, [Addr])	s=0	abcd	ABCD	YES	s=0	dABC	ABCD	YES
	s=1	bcdA	ABCD	YES	s=1	cdAB	ABCD	YES
	s=2	cdAB	ABCD	YES	s=2	bcdA	ABCD	YES
	s=3	dABC	ABCD	YES	s=3	abcd	ABCD	YES
LCE (LCE rD, [Addr])	s=0	abcd	(Not Set)	NO	s=0	dABC	ABCD	YES
	s=1	bcdA	ABCD	YES	s=1	cdAB	ABCD	YES
	s=2	cdAB	ABCD	YES	s=2	bcdA	ABCD	YES
	s=3	dABC	ABCD	YES	s=3	abcd	(Not Set)	NO

图 2.3 装载合并指令操作说明

lcw——装载合并字数据

指令格式：lcw rD, [rA]+

lcw 指令将存储器中指定地址的数据装载入特殊寄存器 LCR（Load Combine Register）中，操作地址由 rA&0xfffffffc 确定。完成操作后，寄存器 rA 的内容自动增加 4。然后，将装载的数据移位，并根据处理器 Endian 模式和 rA 地址的最低有效 2 位将移位结果与 LCR 寄存器中的原始值合并，以实现非对齐的存储器数据的存取。由于在对存储器进行存取操作时 rA 的最低两位被截去，所以不会发生地址对齐错误。该指令与 lcb、lce 配合可以完成对非对齐地址数据的存取。

lce——装载合并字数据结束

指令格式：lce rD, [rA]+

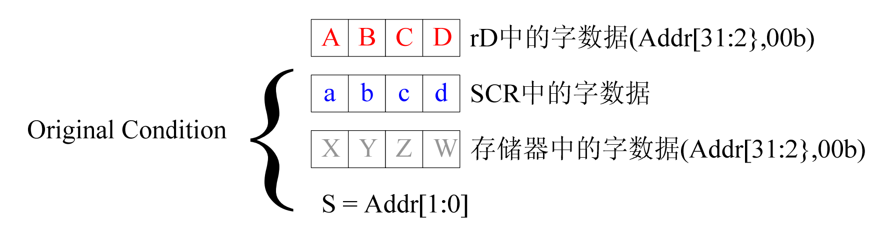
lce 指令将存储器中指定地址的数据装载入特殊寄存器 LCR（Load Combine Register）中，操作地址由 rA&0xfffffffc 确定。完成操作后，寄存器 rA 的内容自动增加 4。然后，将装载的数据移位，并根据处理器 Endian 模式和 rA 地址的最低有效 2 位将移位结果与 LCR 寄存器中的原始值合并，以实现非对齐的存储器数据的存取。若 rA 的最低有效 2 位为 0，则特殊寄存器 LCR 在 lce 指令执行后不会改变。由于在对存储器进行存取操作时 rA 的最低两位被截去，所以不会发生地址对齐错误。该指令与 lcb、lcw 配合可以完成对非对齐地址数据的存取。

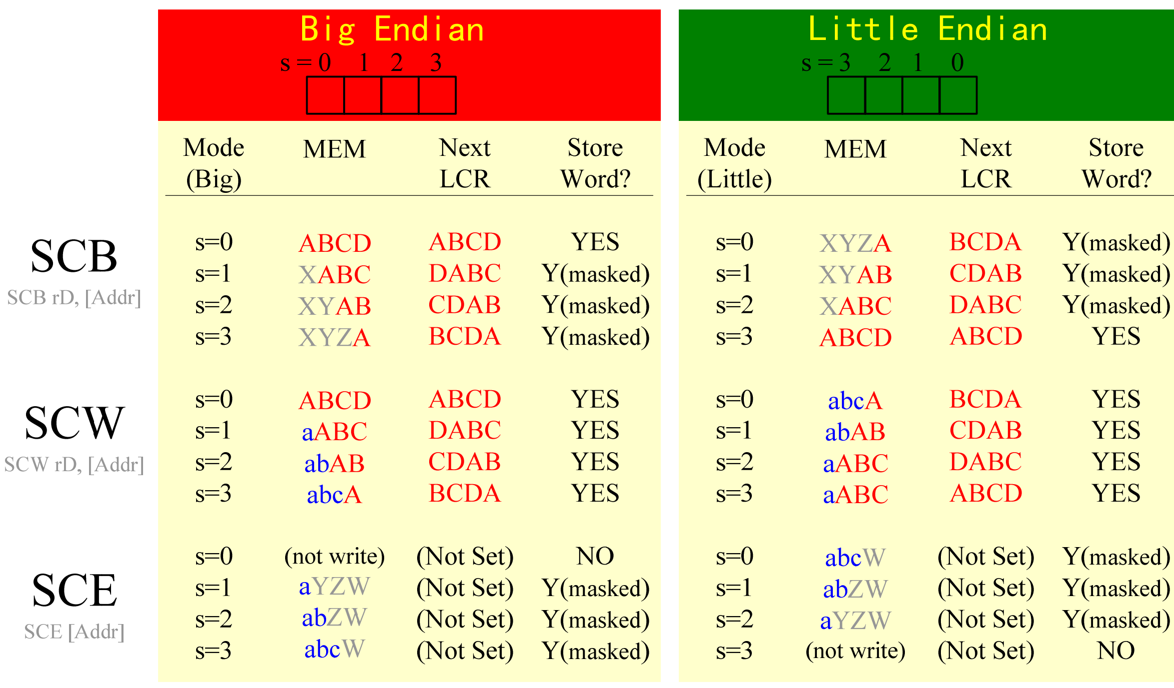
scb——存储合并字数据开始

指令格式：scb rD, [rA]+

scb 指令将存储器中的字数据与通用寄存器 rD 中的数据合并到特殊寄存器 SCR（Store Combine Register）中，然后将合并结果存储到存储器中，该存储地址由 rA&0xfffffffc 确定。存储操作完成后，寄存器 rA 内容自动增加 4。存储数据量和地址取决于处理器 Endian 模式和寄存器 rA 中最低有效 2 位。由于在对存储器进行存取操作时 rA 的最低两位被截去，所以不会发生地址对齐错误。该指令与 scw、sce 配合可以完成对非对齐地址数据的存取。

图 2.4 说明了利用特殊寄存器 SCR 将存储器中的字数据与寄存器 rD 的内容合并并存储的操作。





	Big Endian s=0 1 2 3				Little Endian s=3 2 1 0			
	Mode (Big)	MEM	Next LCR	Store Word?	Mode (Little)	MEM	Next LCR	Store Word?
SCB SCB rD, [Addr]	s=0	ABCD	ABCD	YES	s=0	XYZA	BCDA	Y(masked)
	s=1	XABC	DABC	Y(masked)	s=1	XYAB	CDAB	Y(masked)
	s=2	XYAB	CDAB	Y(masked)	s=2	XABC	DABC	Y(masked)
	s=3	XYZA	BCDA	Y(masked)	s=3	ABCD	ABCD	YES
SCW SCW rD, [Addr]	s=0	ABCD	ABCD	YES	s=0	abcA	BCDA	YES
	s=1	aABC	DABC	YES	s=1	abAB	CDAB	YES
	s=2	abAB	CDAB	YES	s=2	aABC	DABC	YES
	s=3	abcA	BCDA	YES	s=3	aABC	ABCD	YES
SCE SCE [Addr]	s=0	(not write)	(Not Set)	NO	s=0	abcW	(Not Set)	Y(masked)
	s=1	aYZW	(Not Set)	Y(masked)	s=1	abZW	(Not Set)	Y(masked)
	s=2	abZW	(Not Set)	Y(masked)	s=2	aYZW	(Not Set)	Y(masked)
	s=3	abcW	(Not Set)	Y(masked)	s=3	(not write)	(Not Set)	NO

图 2.4 存储合并指令操作说明

scw——装载合并字数据

指令格式：scw rD, [rA]+

scw 指令将合并到特殊寄存器 SCR 中的字数据存储到存储器中，存储地址由 rA&0xfffffffc 确定。完成操作后，寄存器 rA 的内容自动增加 4。然后，源寄存器 rD 和 SCR 寄存器会根据处理器 Endian 模式和寄存器 rA 的最低有效 2 位进行数据合并，而存储操作总是以字为单位。由于在对存储器进行存取操作时 rA 的最低两位被截去，所以不会发生地址对齐错误。该指令与 scb、sce 配合可以完

成对非对齐地址数据的存取。

sce——装载合并字数据结束

指令格式：sce [rA]+

sce 指令将特殊寄存器 SCR 中的数据存储到存储器中，存储地址由 rA&0xfffffffc 确定。完成操作后，寄存器 rA 的内容自动增加 4。存储数据量和地址取决于处理器 Endian 模式和寄存器 rA 中最低有效 2 位。由于在对存储器进行存取操作时 rA 的最低两位被截去，所以不会发生地址对齐错误。该指令与 scb、scw 配合可以完成对非对齐地址数据的存取。

2.3.2 数据处理指令

S+core 处理器的数据处理指令大致可以分为六类：算术运算指令、逻辑运算指令、位运算指令、数据传送指令、移位运算指令和扩展操作指令。大多数的数据处理指令都可以选择.c 后缀，表示是否需要影响条件标志位参与运算。表 2.5 所示为 S+core 的数据处理指令表。

表 2.5 S+core 数据处理指令表

助记符	说明	格式
add{.c}	加法运算	add{.c} rD, rA, rB
addc{.c}	带进位加法运算	addc{.c} rD, rA, rB
addi{.c}	立即数加法运算	addi{.c} rD, SImm16
addis{.c}	移位立即数加法运算	addis{.c} rD, SImm16
addri{.c}	寄存器立即数加法运算	addri{.c} rD, rA, SImm14
sub{.c}	减法运算	sub{.c} rD, rA, rB
subc{.c}	带借位减法运算	subc{.c} rD, rA, rB
subi{.c}	立即数减法运算	subi{.c} rD, SImm16
subis{.c}	移位立即数减法运算	subis{.c} rD, SImm16
subri{.c}	寄存器立即数减法运算	subri{.c} rD, rA, SImm14
neg{.c}	取负	neg{.c} rD, rB
cmp{TCS}.c	比较	cmp{TCS}.c rA, rB
cmpi.c	立即数比较	cmpi.c rA, SImm16
cmpz{TCS}.c	零比较	cmpz{TCS}.c rA
mul	乘法运算	mul rA, rB
mulu	无符号数乘法运算	mulu rA, rB

助记符	说明	格式
div	除法运算	div rA, rB
divu	无符号数除法运算	divu rA, rB
and{.c}	逻辑与	and{.c} rD, rA, rB
andi{.c}	立即数逻辑与	andi{.c} rD, Imm16
andis{.c}	移位立即数逻辑与	andis{.c} rD, Imm16
andri{.c}	寄存器立即数逻辑与	andri{.c} rD, rA, Imm14
or{.c}	逻辑或	or{.c} rD, rA, rB
ori{.c}	立即数逻辑或	ori{.c} rD, Imm16
oris{.c}	移位立即数逻辑或	oris{.c} rD, Imm16
orri{.c}	寄存器立即数逻辑或	orri{.c} rD, rA, Imm14
xor{.c}	逻辑异或	xor{.c} rD, rA, rB
not{.c}	逻辑异非	not{.c} rD, rA
t{cond}	测试并置 T 条件标志	t{cond}
bittst.c	位测试	bittst.c rD, BN
bitset.c	位置位	bitset.c rD, rA, BN
bitclr.c	位清零	bitclr.c rD, rA, BN
bittgl.c	位翻转	bittgl.c rD, rA, BN
mv{cond}	数据传送（有条件）	mv{cond} rD, rA
mtcr	传送数据到控制寄存器	mtcr rD, Crn
mfcr	从控制寄存器传送数据	mfcr rD, Crn
rol{.c}	循环左移	rol{.c} rD, rA, rB
rolc.c	带进位循环左移	rolc.c rD, rA, rB
roli{.c}	立即数循环左移	roli{.c} rD, rA, Imm5
rolic.c	立即数带进位循环左移	rolic.c rD, rA, Imm5
ror{.c}	循环右移	ror{.c} rD, rA, rB
rorc.c	带进位循环右移	rorc.c rD, rA, rB
rori{.c}	立即数循环右移	rori{.c} rD, rA, Imm5

助记符	说明	格式
roric.c	立即数带进位循环右移	roric.c rD, rA, Imm5
sll{.c}	逻辑左移	sll{.c} rD, rA, rB
slli{.c}	立即数逻辑左移	slli{.c} rD, rA, Imm5
sra{.c}	算术右移	sra{.c} rD, rA, rB
srai{.c}	立即数算术右移	srai{.c} rD, rA, Imm5
srl{.c}	逻辑右移	srl{.c} rD, rA, rB
srli{.c}	立即数逻辑右移	srli{.c} rD, rA, Imm5
extsb{.c}	扩展有符号字节数据	extsb{.c} rD, rA
extzb{.c}	扩展无符号字节数据	extzb{.c} rD, rA
extsh{.c}	扩展有符号半字数据	extsh{.c} rD, rA
extzh{.c}	扩展无符号半字数据	extzh{.c} rD, rA

一、算术运算指令

add——加法运算

指令格式：add{extend}{.c} rD, operand2

add 指令可以完成寄存器内容或寄存器与立即数之间的加法运算。其中，.c 为可选，带有.c 的指令在操作完毕后将影响条件标志位的状态，否则将不影响其状态；第二个操作数 operand2 由寄存器或立即数构成；extend 为附加的运算说明，可选值有：

空：表示不带进位的加法，operand2 由两个寄存器组成。指令格式为：add{.c} rD, rA, rB，完成 rD=rA+rB 的操作；

"c"：表示带进位的加法，operand2 由两个寄存器组成。指令格式为：addc{.c} rD, rA, rB，完成 rD=rA+rB+c 的操作；

"i"：表示与立即数进行加法运算，operand2 由一个立即数组成，指令格式为：addi{.c} rD, SImm16，完成 rD=rD+SImm16 的操作；

"is"：表示与移位立即数进行加法运算，operand2 由一个立即数组成，指令格式为：addis{.c} rD, SImm16，完成 rD=rD+(SImm16<<16)的操作；

"ri"：表示寄存器立即数加法运算，operand2 由一个寄存器和一个立即数组成，指令格式为：addri{.c} rD, rA, SImm14，完成 rD=rA+SImm14 的操作。

指令举例如下：

add r4, r2, r1 // r4=r2+r1，不影响条件标志位

add.c r4, r2, r1　　　　// r4=r2+r1，影响条件标志位

addc r4, r2, r1　　　　// r4=r2+r1+c，不影响条件标志位

addis r4, 0x1234　　　　// r4=r4+0x12340000，不影响条件标志位

addri r4, r2, 0x123　　　　// r4=r2+0x123，不影响条件标志位

sub——加法运算

指令格式：sub{extend}{.c} rD, operand2

sub 指令可以完成寄存器内容或寄存器与立即数之间的减法运算。其中，extend 为附加的运算说明，可选值有：

空：表示不带借位的减法，operand2 由两个寄存器组成，指令格式为：sub{.c} rD, rA, rB，完成 rD=rA-rB 的操作；

"c"：表示带借位的加法，operand2 由两个寄存器组成，指令格式为：subc{.c} rD, rA, rB，完成 rD=rA-rB-c 的操作；

"i"：表示与立即数进行减法运算，operand2 由一个立即数组成，指令格式为：subi{.c} rD, SImm16，完成 rD=rD-SImm16 的操作；

"is"：表示与移位立即数进行减法运算，operand2 由一个立即数组成，指令格式为：subis{.c} rD, SImm16，完成 rD=rD-(SImm16<<16)的操作；

"ri"：表示寄存器立即数加法运算，operand2 由一个寄存器和一个立即数组成，指令格式为：subri{.c} rD, rA, SImm14，完成 rD=rA-SImm14 的操作。

指令举例如下：

sub r4, r2, r1　　　　// r4=r2-r1，不影响条件标志位

sub.c r4, r2, r1　　　　// r4=r2-r1，影响条件标志位

subc r4, r2, r1　　　　// r4=r2-r1-c，不影响条件标志位

subis r4, 0x1234　　　　// r4=r4-0x12340000，不影响条件标志位

subir r4, r2, 0x123　　　　// r4=r2-0x123，不影响条件标志位

neg——取负运算

指令格式：neg{.c} rD, rB

neg 指令计算 0-rB 的值，并保存至 rD 寄存器内，完成对寄存器内容的求负（取补）运算。

指令举例如下：

neg r4, r2　　　　// r4=0-r2

cmp——比较运算

指令格式：cmp{TCS}.c rA, rB

cmp 指令进行 rA-rB 的操作，并更新条件标志位，但不会更改任何通用寄存器的内容。

与 add 等指令不同的是，cmp 指令必须带有.c 字段，表示比较完成后更新条件标志位；{TCS} 字段用来指定是否需要更新 T 标志位，在实际的指令编码中，TCS 字段占 2 位。其编码意义如表 2.6 所示。

表 2.6 比较指令的 TCS 字段编码意义表

	命令	TCS		操作
0	CMPTEQ.c	0	0	比较，若 Z=1 则置位 T，否则清 T
1	CMPTMI.c	0	1	比较，若 N=1 则置位 T，否则清 T
2	-	1	0	保留（类似于 CMP.c）
3	CMP.c	1	1	比较

指令举例如下：

```
cmp.c r4, r2                    // r4-r2，根据结果更新条件标志位
```

mul/mulu——乘法运算

指令格式：mul rA, rB

mul 指令对通用寄存器 rA 和 rB 做有符号乘法运算。被乘数为 rA 寄存器内的有符号值，乘数为 rB 寄存器内的有符号值，乘积的低字被保存在 Custom Engine 寄存器 CEL 中，高字保存在 CEH 中。

mulu 指令是对两个无符号字数据做乘法运算，运算结果同样保存在 CEL 和 CEH 中。

指令举例如下：

```
mul r4, r2
mulu r4, r2
```

div/divu——除法运算

指令格式：div rA, rB

div 指令对通用寄存器 rA 和 rB 做有符号除法运算。被除数为 rA 寄存器内的有符号值，除数为 rB 寄存器内的有符号值，除法运算的商被保存在 Custom Engine 寄存器 CEL 中，余数保存在 CEH 中。如果除数为 0，则会引起 Custom Engine 执行异常，此时的操作结果是不可预期的。

divu 指令是对两个无符号字数据做除法运算，运算结果同样保存在 CEL 和 CEH 中。

指令举例如下：

```
div r4, r2
divu r4, r2
```

二、逻辑运算指令

and——与运算

指令格式：and{extend}{.c} rD, operand2

and 指令可以以位操作的方式完成寄存器内容或寄存器与立即数之间的与运算。其中，extend 为附加的运算说明，extend 的可选值有：

空：表示寄存器之间的与运算，operand2 由两个寄存器组成，指令格式为：and{.c} rD, rA, rB，完成 rD=rA&rB 的操作；

“i”：表示与立即数进行与运算，operand2 由一个立即数组成，指令格式为：andi{.c} rD, Imm16, 完成 rD=rD&Imm16 的操作；

“is”：表示与移位立即数进行与运算，operand2 由一个立即数组成，指令格式为：andis{.c} rD, Imm16，完成 rD=rD&(Imm16<<16)的操作；

“ri”：表示寄存器与立即数进行与运算，operand2 由一个寄存器和一个立即数组成，指令格式为：andri{.c} rD, rA, Imm14，完成 rD=rA&Imm14 的操作。

指令举例如下：

```
and r4, r2, r1              // r4=r2&r1，不影响条件标志位
and.c r4, r2, r1            // r4=r2&r1，影响条件标志位
andis r4, 0x1234            // r4=r4&0x12340000，不影响条件标志位
andri r4, r2, 0x123         // r4=r2&0x123，不影响条件标志位
```

or——或运算

指令格式：or{extend}{.c} rD, operand2

or 指令可以以位操作的方式完成寄存器内容或寄存器与立即数之间的或运算。其中，extend 为附加的运算说明，可选值有：

空：表示寄存器之间的或运算，operand2 由两个寄存器组成，指令格式为：or{.c} rD, rA, rB，完成 rD=rA | rB 的操作；

“i”：表示与立即数进行或运算，operand2 由一个立即数组成，指令格式为：ori{.c} rD, Imm16, 完成 rD=rD|Imm16 的操作；

“is”：表示与移位立即数进行或运算，operand2 由一个立即数组成，指令格式为：oris{.c} rD, Imm16，完成 rD=rD|(Imm16<<16)的操作；

“ri”：表示寄存器与立即数进行或运算，operand2 由一个寄存器和一个立即数组成，指令格式为：orri{.c} rD, rA, Imm14，完成 rD=rA|Imm14 的操作。

指令举例如下：

```
or r4, r2, r1               // r4=r2 | r1，不影响条件标志位
```

or.c r4, r2, r1　　　　　　// r4=r2 | r1，影响条件标志位

oris r4, 0x1234　　　　　　// r4=r4 | 0x12340000，不影响条件标志位

orri r4, r2, 0x123　　　　　// r4=r2 | 0x123，不影响条件标志位

xor——异或运算

指令格式：xor{.c} rD, rA, rB

xor 指令可以以位操作的方式将 rA 和 rB 的内容进行异或运算并将结果保存至 rD 寄存器内。

指令举例如下：

xor r4, r2, r3　　　　　　// r4 = r2 xor r3，不影响条件标志位

not——取反运算

指令格式：not{.c} rD, rA

not 指令可以以位操作的方式将 rA 的内容进行取反并将结果保存至 rD 寄存器内。指令格式如下：

指令举例如下：

not r4, r2,　　　　　　// r4 = (r2 按位取反)，不影响条件标志位

t——测试并置 T 标志位

指令格式：t{cond}

t 指令用于对 T 标志进行操作。该 T 标志用于并行条件执行（PCE）指令中。根据 T 的值可以判断 PCE 指令要执行哪个指令分支。其中，cond 后缀为 EC（EXECUTE CONDITION）字段的条件标志位测试条件（如表 2.7 所示），如果测试条件为真，则置位 T 标志，否则，清零 T 标志。

表 2.7 EC 字段的 cond 编码意义表

序号	cond 后缀	操作	测试
0	cs	当 C 为 1 时执行操作	C
1	cc	当 C 为 0 时执行操作	~C
2	gtu	当 C 为 1 且 Z 为 0 时执行操作	C & ~Z
3	leu	当 C 为 0 或 Z 为 1 时执行操作	~C | Z
4	eq	当 Z 为 1 时执行操作	Z
5	ne	当 Z 为 0 时执行操作	~Z
6	gt	当 Z 为 0 且 N 为 V 时执行操作	(Z=0)&(N=V)

序号	cond 后缀	操作	测试
7	le	当 Z 为 1 且 N 不为 V 时执行操作	(Z=1)&(N!=V)
8	ge	当 N 为 V 时执行操作	N=V
9	lt	当 N 不为 V 时执行操作	V!=V
10	mi	当 N 为 1 时执行操作	N
11	pl	当 N 为 0 时执行操作	~N
12	vs	当 V 为 1 时执行操作	V
13	vc	当 V 为 0 时执行操作	~V
14	-	空操作	-
15	al	无条件执行操作	-

指令举例如下：

tge　　　　　　　// 当 N 为 0 时置位 T 标志

三、位运算指令

bittst.c——寄存器位测试

指令格式：bittst.c rA, BN

bittst 指令测试通用寄存器 rA 的第 BN 位，并更新 Z 标志。同时，N 标志也将受到影响。

指令举例如下：

bittst r4, 10　　　　　　// 测试 r4 寄存器的 bit10 位

bitset.c——寄存器位置 1

指令格式：bitset.c rD, rA, BN

bitset 指令可以将通用寄存器 rA 的第 BN 位置 1，并更新 Z 标志。同时，N 标志也将受到影响。

指令举例如下：

bitset.c r4, r5, 10　　　　　　// 将 r5 寄存器的 bit10 位置 1，并将结果保存至 r4

bitclr.c——寄存器位置 0

指令格式：bitclr.c rD, rA, BN

bitclr 指令可以将通用寄存器 rA 的第 BN 位置 0，并更新 Z 标志。同时，N 标志也将受到影响。

指令举例如下：

bitclr.c r4, r5, 10　　　　　　// 将 r5 寄存器的 bit10 位置 0，并将结果保存至 r4

bittgl.c——寄存器位取反

指令格式：bittgl.c rD, rA, BN

bittgl 指令可以将通用寄存器 rA 的第 BN 位取反，并更新 Z 标志。同时，N 标志也将受到影响。

指令举例如下：

```
bittgl.c r4, r5, 10          // 将 r5 寄存器的 bit10 位取反，并将结果保存至 r4
```

四、数据传送指令

mv{cond}——条件数据传输

指令格式：mv{cond} rD, rA

mv 指令根据对条件标志位的测试结果将 rA 寄存器的内容传送至 rD 寄存器。

其中，cond 后缀为 EC（EXECUTE CONDITION）字段的条件标志位测试条件（如表 2.7 所示），如果测试条件为真，则执行数据传输操作，否则 rD 寄存器的内容不作改变。

指令举例如下：

```
ldi r4, 0x100
cmp r2, r4                   // 比较 r2 与 0x100 的大小
mvge r4, r2                  // 若 r2>=0x100，则将 r2 的内容传输到 r4
```

mtcr——传输数据到控制寄存器

指令格式：mtcr rD, Crn

mtcr 指令可以将通用寄存器 rD 的内容传输给控制寄存器 Crn，n 表示控制寄存器编号。通常该指令只能用于核心模式，但如果将 PSR 寄存器中的 cra 位置 1，则可以在用户模式下使用该指令。

指令举例如下：

```
mtcr r4, Cr16                // 将 r4 的内容传输给 LDM 物理帧数寄存器 Cr16
```

mfcr——从控制寄存器传输数据

指令格式：mfcr rD, Crn

mfcr 指令可以将控制寄存器 Crn 的内容传输给通用寄存器 rD，n 表示控制寄存器编号。与 mtcr 类似，通常该指令只能用于核心模式，但如果将 PSR 寄存器中的 cra 位置 1，则可以在用户模式下使用该指令。

指令举例如下：

```
mfcr r4, Cr16                // 将 LDM 物理帧数寄存器 Cr16 的内容传输给通用寄存器 r4
```

五、移位运算指令

rol——循环左移

指令格式：rol{i}{c}{.c} rD, rA, operand2

rol 指令将 rA 寄存器的内容向左循环移动 operand2 指定的位数，并存放在 rD 寄存器内。其中，带后缀“i”表示移动的位数 operand2 由一个 5 位的立即数指定，否则由一个通用寄存器的最低 5 位指定；带后缀“c”表示带进位的循环移位，否则为不带进位的循环移位。

需要注意的是，带进位的循环移位指令必须带有.c 后缀，因为此时 C 标志位将受到影响。

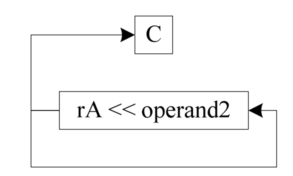


图 2.5 不带进位的循环左移

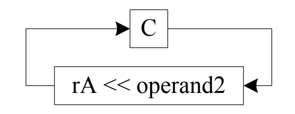


图 2.6 带进位的循环左移

指令举例如下：

```
rol r4, r2, r3            // 不带进位的循环左移，移动位数由寄存器 r3 决定
rolc.c r4, r2, r3         // 带进位的循环左移，移动位数由寄存器 r3 决定，
                          // 影响条件标志位
roli r4, r2, 0x0a         // 不带进位的循环左移，移动位数由立即数 0x0a 决定
rolic.c r4, r2, 0x03      // 带进位的循环左移，移动位数由立即数 0x03 决定
```

ror——循环右移

指令格式：ror{i}{c}{.c} rD, rA, operand2

ror 指令将 rA 寄存器的内容向右循环移动 operand2 指定的位数，并存放在 rD 寄存器内。其中，带后缀“i”表示移动的位数 operand2 由一个 5 位的立即数指定，否则由一个通用寄存器的最低 5 位指定；带后缀“c”表示带进位的循环移位，否则为不带进位的循环移位。

需要注意的是，带进位的循环移位指令必须带有.c 后缀，因为此时 C 标志位将受到影响。

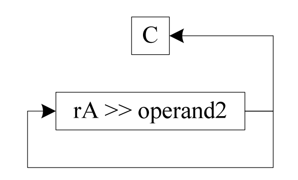


图 2.7 不带进位的循环右移

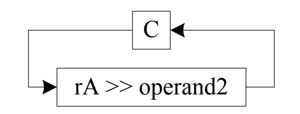


图 2.8 带进位的循环右移

指令举例如下：

```
ror r4, r2, r3          // 不带进位的循环右移，移动位数由寄存器 r3 决定
rorc.c r4, r2, r3       // 带进位的循环右移，移动位数由寄存器 r3 决定，
                        // 影响条件标志位
rori r4, r2, 0x0a       // 不带进位的循环右移，移动位数由立即数 0x0a 决定
roric.c r4, r2, 0x03    // 带进位的循环右移，移动位数由立即数 0x03 决定
```

sll——逻辑左移

指令格式：sll{i}{.c} rD, rA, operand2

sll 指令将 rA 寄存器的内容向左移动 operand2 指定的位数，并存放在 rD 寄存器内。其中，带后缀“i”表示移动的位数 operand2 由一个 5 位的立即数指定，否则由一个通用寄存器的最低 5 位指定。

指令举例如下：

```
sll r4, r2, r3          // 逻辑左移，移动位数由寄存器 r3 决定
slli r4, r2, 0x0a       // 逻辑左移，移动位数由立即数 0x0a 决定
```

srl——逻辑右移

指令格式：srl{i}{.c} rD, rA, operand2

srl 指令将 rA 寄存器的内容向右移动 operand2 指定的位数，并存放在 rD 寄存器内。其中，带后缀“i”表示移动的位数 operand2 由一个 5 位的立即数指定，否则由一个通用寄存器的最低 5 位指定。

指令举例如下：

```
srl r4, r2, r3          // 逻辑右移，移动位数由寄存器 r3 决定
```

```
srli r4, r2, 0x0a                    // 逻辑右移，移动位数由立即数 0x0a 决定
```

sra——算术右移

指令格式：sra{i}{.c} rD, rA, operand2

sra 指令将 rA 寄存器的内容向右移动 operand2 指定的位数，同时用符号位填充左端的空缺，然后存放在 rD 寄存器内。其中，带后缀“i”表示移动的位数 operand2 由一个 5 位的立即数指定，否则由一个通用寄存器的最低 5 位指定。

指令举例如下：

```
sra r4, r2, r3          // 算术右移，移动位数由寄存器 r3 决定
srai r4, r2, 0x0a       // 算术右移，移动位数由立即数 0x0a 决定
```

六、扩展操作指令

扩展操作指令可以将字节数据或半字数据的高字节进行符号填充或零填充而形成字数据。

extsb——扩展有符号字节数据

指令格式：extsb{.c} rD, rA

extsb 指令将 rA 寄存器的低字节进行符号扩展，并将结果保存在 rD 寄存器内。执行该操作后，rD 寄存器的高三个字节全部填充为 rA 寄存器的最低字节的符号位（bit7 位），如图 2.9 所示。

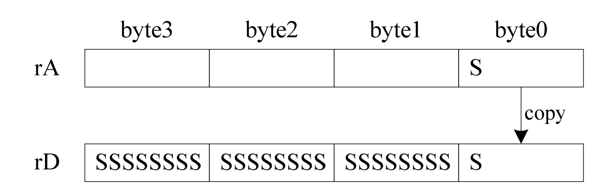


图 2.9 扩展有符号字节数据示意图

指令举例如下：

```
ldi r2, 0x80
extsb r4, r2            // 执行该指令后 r4 的内容为 0xffffff80
ldi r2, 0x70
extsb.c r4, r2          // 执行该指令后 r4 的内容为 0x00000070，条件标志位相应改变
```

extzb——扩展无符号字节数据

指令格式：extzb{.c} rD, rA

extzb 指令将 rA 寄存器的低字节进行零扩展，并将结果保存在 rD 寄存器内。执行该操作后，rD 寄存器的高三个字节全部被填充为 0，如图 2.10 所示。

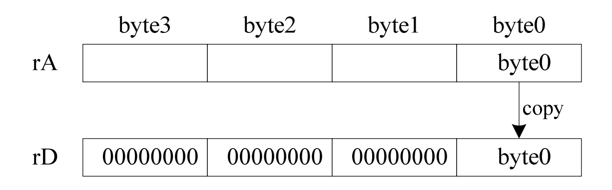


图 2.10 扩展无符号字节数据示意图

指令举例如下：

```
ldi r2, 0x80
extzb r4, r2            // 执行该指令后 r4 的内容为 0x00000080
```

ldi r2, 0x70

extzb.c r4, r2　　　　　　// 执行该指令后 r4 的内容为 0x00000070，条件标志位相应改变

extsh——扩展有符号半字数据

指令格式：extsh{.c} rD, rA

extsh 指令将 rA 寄存器的低半字进行符号扩展，并将结果保存在 rD 寄存器内。执行该操作后，rD 寄存器的高半字全部填充为 rA 寄存器的低半字的符号位（bit15 位），如图 2.11 所示。

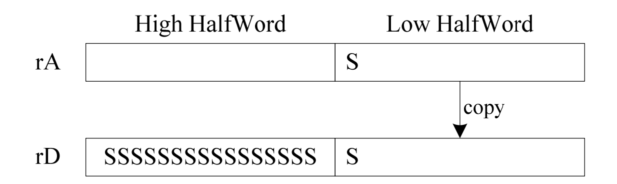


图 2.11 扩展有符号半字数据示意图

指令举例如下：

ldi r2, 0x8000

extsh r4, r2　　　　　　// 执行该指令后 r4 的内容为 0xffff8000

ldi r2, 0x7000

extsh.c r4, r2　　　　　　// 执行该指令后 r4 的内容为 0x00007000，条件标志位相应改变

extzh——扩展无符号半字数据

指令格式：extzh{.c} rD, rA

extzh 指令将 rA 寄存器的低半字进行零扩展，并将结果保存在 rD 寄存器内。执行该操作后，rD 寄存器的高半字全部被填充为 0，如图 2.12 所示。

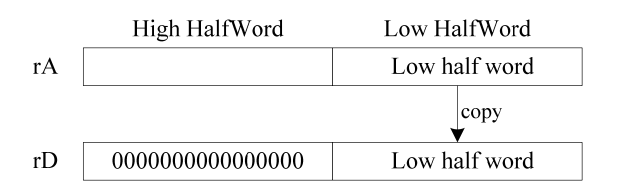


图 2.12 扩展无符号半字数据示意图

指令举例如下：

ldi r2, 0x8000

extzh r4, r2　　　　　　// 执行该指令后 r4 的内容为 0x00008000

ldi r2, 0x7000

extzh.c r4, r2　　　　　　// 执行该指令后 r4 的内容为 0x00007000，条件标志位相应改变

2.3.3　分支指令

分支指令用来实现程序的跳转。表 2.8 所示为 S+core 的分支指令表。其中，后缀“l”表示带链接的跳转，带有该后缀后在跳转前将把本条指令后的下一条指令的地址置入链接寄存器 r3 中。

表 2.8 S+core 分支指令表

助记符	说明	格式
j	无条件跳转	j{l} label
b	条件分支	b{cond}{l} label
br	条件寄存器分支	br{cond}{l} rA

j——无条件跳转

指令格式：j{l} label

j 指令跳转到制定地址处执行程序，并限制跳转范围为当前指令的±16MB 范围内。其中 label 是具有 24 位有效位数的偏移地址。目标地址是这样形成的：先将 24 位偏移地址左移一位，然后在与本条指令的高 7 位结合，形成目标地址。

指令举例如下：

```
j label                // 跳转至 label 标号处
jl label               // 跳转至 label 标号处并保存下一条指令地址，可以用于函数调用
```

b——条件分支

指令格式：b{cond}{l} label

b 指令根据 cond 所代表的 BC（Branch Condition）字段的条件标志位测试结果分支到目标地址。其中 label 是具有 19 位有效位数的分支偏移量。目标地址为分支偏移量左移一位并作符号扩展与当前 PC 内容的地址之和。

BC（Branch Condition）字段的 cond 编码如表 2.9 所示。

表 2.9 BC 字段的 cond 编码意义表

序号	cond 后缀	操作	测试
0	cs{l}	当 C 为 1 时执行操作	C
1	cc{l}	当 C 为 0 时执行操作	~C
2	gtu{l}	当 C 为 1 且 Z 为 0 时执行操作	C & ~Z
3	leu{l}	当 C 为 0 或 Z 为 1 时执行操作	~C | Z
4	eq{l}	当 Z 为 1 时执行操作	Z

序号	cond 后缀	操作	测试
5	ne{l}	当 Z 为 0 时执行操作	~Z
6	gt{l}	当 Z 为 0 且 N 为 V 时执行操作	(Z=0)&(N = V)
7	le{l}	当 Z 为 1 且 N 不为 V 时执行操作	(Z=1)&(N != V)
8	ge{l}	当 N 为 V 时执行操作	N=V
9	lt{l}	当 N 不为 V 时执行操作	V!=V
10	mi{l}	当 N 为 1 时执行操作	N
11	pl{l}	当 N 为 0 时执行操作	~N
12	vs{l}	当 V 为 1 时执行操作	V
13	vc{l}	当 V 为 0 时执行操作	~V
14	cnz{l}	CNT>0 时执行操作	CNT>0
15	al{l}	无条件执行操作	-

指令举例如下：

```
beq label          // Z=0 时分支至 label 标号处
bgel label         // N 为 V 时分支至 label 标号处并保存下一条指令地址
                   //  可以用于函数调用
```

br——条件寄存器分支

指令格式：br{cond}{l} rA

br 指令根据对条件标志位的测试结果分支到由 rA 寄存器指定的目标地址。

分支条件 cond 如表 2.9 所示。

指令举例如下：

```
breq r3            // Z=0 时分支至 r3 指定的地址处
brgel r3           // N 为 V 时分支至 r3 指定的地址处并保存下一条指令地址
                   //  可以用于函数调用
```

2.3.4 特殊指令

S+core 处理器具有以下几种特殊指令：Custom Engine 指令、特殊寄存器控制指令、Cache 控制指令、系统控制指令等。表 2.10 所示为 S+core 特殊指令表：

表 2.10 S+core 特殊指令表

助记符	说明	格式
ceinst	Custom Engine 用户定义	ceinst func5, rA, rB, USD1, USD2
mtcex	传送数据到 Custom Engine 寄存器	mtcel rD mtceh rD mtcehl rD, rA
mfcex	从 Custom Engine 寄存器传送数据到通用寄存器	mfcel rD mfceh rD mfcehl rD, rA
mtsr	传送数据到特殊寄存器	mtsr rA, Srn
mfsr	从特殊寄存器传送数据到通用寄存器	mfsr rD, Srn
cache	cache 控制指令	cache cache_op, [rA, SImm15]
trap	条件陷阱	trap{cond} Software_Parameter(Imm5)
syscall	系统调用	syscall Software_Parameter(Imm15)
sleep	睡眠	sleep
sdbbp	软件 Debug 断点	sdbbp code(Imm5)
pflush	清空流水线	pflush
rte	从异常返回	rte
drte	从 debug 异常返回	drte

Custom Engine 指令

Custom Engine 指令主要用于 Custom Engine 指令扩展以及对 Custom Engine 寄存器 CEH 和 CEL 的操作。

ceinst——Custom Engine 用户定义

指令格式：ceinst func5, rA, rB, USD1, USD2

ceinst 格式的指令是为了实现 custom engine 指令扩展而定义的。其中的 func5 定义了具体的 Custom Engine 操作，且该格式指令允许至多用两个通用寄存器作为源操作数。在该指令中定义了 rA 和 rB 字段，若需要指定一或者两个源通用寄存器，则相应寄存器索引值就会被置于这两个字段内。USD1 和 USD2 为用户自定字段，这两个字段既可以是用于计算的立即数，亦可以是用于操作控制的参数，或者是目标通用寄存器索引号。若 Custom Engine 不执行这类扩展指令，则会发生未定义指令异常。

指令举例如下：

ceinst 0x00, r2, r4, 0x01, 0x02

mtcex——传送数据到 Custom Engine 寄存器

指令格式： mtcel rD

mtceh rD

mtcehl rD, rA

mtcex 指令可以将通用寄存器的内容写入 Custom Engine 寄存器 CEL 或 CEH 内。mtcel 指令将 rD 寄存器内容写入 CEL；mtceh 指令将 rD 寄存器内容写入 CEH；mtcehl 指令将 rD 寄存器内容写入 CEH，并将 rA 寄存器内容写入 CEL。

指令举例如下：

mtcel r4 // 将 r4 寄存器内容传递给 CEL

mfcex——从 Custom Engine 寄存器传送数据到通用寄存器

指令格式： mfcel rD

mfceh rD

mfcehl rD, rA

mfcex 指令可以将 Custom Engine 寄存器 CEL 或 CEH 的内容传递给通用寄存器。mfcel 指令将 CEL 内容写入 rD 寄存器；mfceh 指令将 CEH 内容写入 rD 寄存器；mfcehl 指令将 CEH 内容写入 rD 寄存器，并将 CEL 内容写入 rA 寄存器。

指令举例如下：

mfcel r4 // 将 CEL 内容传递给 r4 寄存器

特殊寄存器控制指令

特殊寄存器控制指令主要完成通用寄存器与特殊寄存器之间的数据传输操作。

mtsr——传送数据到特殊寄存器

指令格式：mtsr rA, Srn

mtsr 指令将通用寄存器 rA 的内容传送给特殊寄存器 Srn（n 为特殊寄存器编号）。

指令举例如下：

mtsr r4, Sr1

mfsr——从特殊寄存器传送数据到通用寄存器

指令格式：mfsr rD, Srn

mfsr 指令将特殊寄存器 Srn（n 为特殊寄存器编号）的内容传送给通用寄存器 rD。

指令举例如下：

mfsr r4, Sr1

Cache 控制指令

cache——Cache 控制指令

指令格式：cache cache_op, [rA, SImm15]

cache 指令根据 cache_op 码进行 Cache 操作。该操作将基地址寄存器 rA 的内容与 15 位立即数 SImm15 相加形成一个有效的虚拟地址，并通过固定映射模式的 MMU 转换成一物理地址。cache_op 操作码对应的操作如表 2.11 所示。

表 2.11 Cache OP 字段编码表

cache_op [4:0]	**I-Cache/ D-Cache**	功能	操作地址
0x00	I-Cache	预取一 Cache 行	rA + SImm15
0x01	I-Cache	预取和锁定一 Cache 行	rA + SImm15
0x02	I-Cache	无效并解锁一 Cache 行	rA + SImm15
0x03	I-Cache	填充 PFN 和 Size 值到 LIM	PFN 由 rA 指定 Size 由 SImm15 指定
0x04	I-Cache	从新填充 PFN 和 Size 值到 LIM	任意
0x08	D-Cache	预取一 Cache 行	rA + SImm15
0x09	D-Cache	预取和锁定一 Cache 行	rA + SImm15
0x0A	D-Cache	无效并解锁一 Cache 行	rA + SImm15
0x0B	D-Cache	填充 PFN 和 Size 值到 LDM	PFN 由 rA 指定 Size 由 SImm15 指定
0x0C	D-Cache	将 LDM 内的 PFN 和 Size 值写回主存储器	PFN 由 rA 指定 Size 由 SImm15 指定
0x0D	D-Cache	强制将一个被改写过的有效 Cache 行中的内容写回主存储器并将该行改写为有效	rA + SImm15
0x0E	D-Cache	强制将一个被改写过的有效 Cache 行中的内容写回主存储器并将改行改写为无效	rA + SImm15
0x10	I-Cache	无效全部 Cache	任意

cache_op [4:0]	I-Cache/ D-Cache	功能	操作地址
0x11	I-Cache	当指令预取缓存功能使能时，使用该指令使指令预取缓存功能禁止； 当指令预取缓存功能禁止时，使用该指令使指令预取缓存功能使能；	任意
0x18	D-Cache	无效全部 Cache	任意
0x1A	D-Cache	清空写缓存器	任意
0x1B	D-Cache	启动写缓存功能	任意
0x1D	D-Cache	启动复写 D-Cache 功能（使能/禁止）	任意
0x1E	D-Cache	强制将所有被改写过的有效 D-Cache 行写回主存储器，并改为有效	任意
0x1F	D-Cache	强制将所有被改写过的有效 D-Cache 行写回主存储器，并改为无效	任意

指令举例如下：

cache 0x00, [r2, 0x0123]

系统控制指令

trap——条件陷阱

指令格式：trap{cond} Software_Parameter(Imm5)

trap 指令是一种控制指令，仅用于核心模式，或在 PSR 寄存器中的 cra 位为 1 时可用于用户模式。trap 指令根据 cond 所代表的 EC（EXECUTE CONDITION）条件标志位的测试结果（见表 2.7）触发陷阱异常。即当测试结果为真时触发陷阱异常，进入相应的异常处理。陷阱异常发生后，只有以下寄存器的值会改变：

- 处理器状态寄存器（PSR）中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；
- 条件寄存器（CR1）的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位
- 异常原因寄存器（ECR）中的 Exc_code 字段码为 10；
- EPC 寄存器指向 trap{cond}指令。

指令举例如下：

trapeq 0x008e

syscall——系统调用陷阱

指令格式：syscall Software_Parameter(Imm15)

syscall 指令可以引发系统调用异常，并进入相应的异常处理程序。该异常发生后，只有以下寄存器的值会改变：

- 处理器状态寄存器（PSR）中的 IEc 与 UMc 位的值被分别压入 IEs 与 UMs 位；
- 条件寄存器（CR1）的 Tc、Nc、Zc、Cc 及 Vc 位的值分别压入 Ts、Ns、Zs、Cs 及 Vs 位
- 异常原因寄存器（ECR）中的 Exc_code 字段码为 7；
- EPC 寄存器指向 syscall 指令。

指令举例如下：

```
syscall 0x008e
```

sleep——睡眠

指令格式：sleep

sleep 指令使处理器内核进入省电待机模式，直至收到外部中断（针对 SPCE3200 芯片 40 个中断源和键唤醒）、非屏蔽中断或 Debug 中断信号为止。执行睡眠指令之前，用户需通过 PSR 寄存器中的 IE 字段和 ECR 寄存器中的 IM 字段将这些中断条件使能。本指令只能用于核心模式，或在 PSR 寄存器中的 cra 位为 1 时可用于用户模式。

sdbbp——软件 Debug 断点

指令格式：sdbbp code(Imm5)

在非调试状态下，sdbbp 指令将引发软件断点 Debug 异常，并进入异常处理程序。该异常的向量地址取决于在线仿真（ICE）和仿真信息获取（Probe）电路。执行本指令后，Debug 寄存器（DREG）中相应的状态位会置位，调试异常程序计数器（DEPC）将指向 sdppb 指令的地址。其中，code 字段指定 5 位软件参数值。

在调试状态下，该指令相当于空操作。

指令举例如下：

```
sdbbp 0x0e
```

pflush——清空流水线

指令格式：pflush

在对于流水线型处理器，有些指令序列需要插入软件 bubble（即 nop 指令），以防止流水线冲突导致数据错误。例如，在庄子合并指令跟随一目标寄存器为 LCR 的 mfsr 指令的情况下，需要 3 个 nop 指令，在这种情况下，也可以插入一个 pflush 指令，来代替若干个 nop 指令，使得软件编程更方便。



pflush 指令的执行类似于跳转到下一条指令，即该指令首先清空下一条指令的流水线状态，再从（pfluash+4）地址获取下一指令继续执行，这很像非流水线处理器的动作。

rte——从异常返回

指令格式：rte

rte 指令用于从中断或异常服务程序子程序返回，其操作类似于寄存器跳转（br）指令。执行完 rte 指令后，程序计数器（PC）的值从 EPC 恢复至中断之前的地址，同时，处理器状态寄存器（PSR）中的 UMc 和 IEc 字段分别恢复自 UMs 和 IEs 字段；条件寄存器中的 Tc、Nc、Zc、Cc 和 Vc 字段分别恢复自 Ts、Ns、Zs、Cs 和 Vs 字段。本指令只能用于核心模式，或在 PSR 寄存器中的 cra 位为 1 时可用于用户模式。

drte——从 Debug 异常返回

指令格式：rte

rte 指令用于从 Debug 异常服务程序子程序返回。执行完 rte 指令后，程序计数器（PC）的值从 DEPC 恢复至中断之前的地址。本指令只能用于核心模式，或在 PSR 寄存器中的 cra 位为 1 时可用于用户模式。

2.3.5 协处理器指令

按照当前 CPU 核心的设计理念，需要尽可能满足各种用户的要求，S+core 为用户留出了这些接口，在设计各种芯片的时候可以挂接最多三个协处理器，辅助完成用户功能。这样设计的好处是，特殊的操作可以交由协处理器完成。

协处理器的控制要通过协处理器指令实现。表 2.12 所示为 S+core 的协处理器指令表。

表 2.12 S+core 协处理器指令表

助记符	说明	格式
COPn	协处理器用户定义（n 为协处理器编号）	COPn cop_code20 或 COPn CrD, CrA, CrB, cop_code5
MTCn	传送数据到协处理器数据寄存器	MTCn rD, CrA
MTCCn	传送数据到协处理器控制寄存器	MTCCn rD, CrA
MFCn	从协处理器数据寄存器传送数据	MFCn rD, CrA
MFCCn	从协处理器控制寄存器传送数据	MFCCn rD, CrA
LDCn	装载数据到协处理器数据寄存器	LDCn CrA, [rD, SImm12]
STCn	从协处理器数据寄存器存储数据	STCn CrA, [rD, SImm12]

COPn——协处理器用户定义

指令格式： COPn cop_code20

或

COPn CrD, CrA, CrB, cop_code5

其中，n 为 1～3，表示协处理器编号；cop_code20 指定 20 位协处理器用户定义码；CrD，CrA，CrB 指定用户定义字段编号；cop_code5 指定 5 位协处理器用户定义码。

该指令是用户定义的协处理器扩展指令，分别有参数为 20 位和参数为 5 位两种类型的格式。需要注意的是，处理器状态寄存器（PSR）中的 CU 字段相应位须使能，否则执行该指令将引起协处理器使用异常。另外，如果没有协处理器响应该指令，则会产生未定义指令异常。

MTCn——传送数据到协处理器数据寄存器

指令格式： MTCn rD, CrA

其中，n 为 1～3，表示协处理器编号。该指令将通用寄存器 rD 的内容传递给指定协处理器的数据寄存器 CrA。需要注意的是，处理器状态寄存器（PSR）中 cu 字段相应位须使能，否则执行该指令将引起协处理器使用异常。

指令举例如下：

mtc3 r14, cr15

MTCCn——传送数据到协处理器控制寄存器

指令格式： MTCCn rD, CrA

其中，n 为 1～3，表示协处理器编号。该指令将通用寄存器 rD 的内容传递给指定协处理器的控制寄存器 CrA。需要注意的是，处理器状态寄存器（PSR）中 cu 字段相应位须使能，否则执行该指令将引起协处理器使用异常。

指令举例如下：

mtcc3 r14, cr15

MFCn——从协处理器数据寄存器传送数据

指令格式： MFCn rD, CrA

其中，n 为 1～3，表示协处理器编号。该指令将指定协处理器的数据寄存器 CrA 的内容传递给通用寄存器 rD。需要注意的是，处理器状态寄存器（PSR）中 cu 字段相应位须使能，否则执行该指令将引起协处理器使用异常。

指令举例如下：

mfc2 r4, cr8

MFCCn——从协处理器控制寄存器传送数据

指令格式： MFCCn rD, CrA

其中，n 为 1～3，表示协处理器编号。该指令将指定协处理器的控制寄存器 CrA 的内容传递给通用寄存器 rD。需要注意的是，处理器状态寄存器（PSR）中 cu 字段相应位须使能，否则执行该指令将引起协处理器使用异常。

指令举例如下：

mfcc3 r4, cr8

LDCn——装载数据到协处理器数据寄存器

指令格式： LDCn CrA, [rD, SImm12]

其中，n 为 1～3，表示协处理器编号。该指令从存储器装载一字数据到指定协处理器的数据寄存器 CrA 中，存储器的操作地址由通用寄存器 rD 的内容与左移 2 位的 10 位有符号数 Imm10 之和指定。注意，存储器操作地址须为字对齐，否则讲发生地址对齐错误异常。另外，处理器状态寄存器（PSR）中 cu 字段相应位须使能，否则执行该指令将引起协处理器使用异常。

指令举例如下：

ldc3 cr8, [r4, 0x0012]

STCn——从协处理器数据寄存器存储数据

指令格式： STCn CrA, [rD, SImm12]

其中，n 为 1～3，表示协处理器编号。该指令将指定协处理器的数据寄存器 CrA 的内容存储到存储器指定地址。存储器的操作地址由通用寄存器 rD 的内容与左移 2 位的 10 位有符号数 Imm10 之和指定。注意，存储器操作地址须为字对齐，否则讲发生地址对齐错误异常。另外，处理器状态寄存器（PSR）中 cu 字段相应位须使能，否则执行该指令将引起协处理器使用异常。

指令举例如下：

stc3 cr8, [r4, 0x0012]

2.4 16 位指令集

S+core 处理器的 16 位指令集可以看作是 32 位指令集的压缩形式的子集，它是专为编译器（compiler）编译混合模式的指令而设计，以减少代码容量提高代码密度，一般不推荐用来进行手动汇编编码。由于 16 位指令集的指令宽度的限制，基本上所有的 16 位指令都是两个操作数的操作，且允许使用小立即数，另外，由于指令长度的限制，指令只能访问 32 个通用寄存器中的前 16 个（r0～r15）。当发生 16 位指令不为字对齐的情况，则最好将该 16 位指令改为相应的 32 位指令，而最好不要插入一个 16 位的 nop!指令来对齐字边界，因为这样将增加指令执行个数。

S+core 处理器支持完全的 16 位/32 位混合编程，两种指令集之间不需要进行任何切换。S+core

处理器的所有 16 位指令的助记符均带有“!”，以和 32 位指令进行区分。大多数 32 位的通用指令加上“!”可以转换成对应的 16 位指令。

2.4.1 装载与存储指令

16 位装载/存储指令仅具有三种基本的寻址方式：立即数寻址、基址寻址和基址变址寻址。另外，还有两种寻址方式：前减量变址寻址和后增量变址寻址，分别用于 push!和 pop!两条指令，可以实现对堆栈的操作。

表 2.13 S+core16 位装载/存储指令表

助记符	说明	格式
lbu{p}!	装载无符号字节数据	lbu! rD_{g0}, $[rA_{g0}]$ 基址寻址 lbup! rDgo, Imm5 基址变址寻址
lh{p}!	装载半字数据	lh! rD_{g0}, $[rA_{g0}]$ 基址寻址 lhp! rD_{g0}, Imm6 基址变址寻址
lw{p}!	装载字数据	lw! rD_{g0}, $[rA_{g0}]$ 基址寻址 lwp! rD_{g0}, Imm7 基址变址寻址
sb{p}!	存储字节数据	sb! rD_{g0}, $[rA_{g0}]$ 基址寻址 sbp! rD_{g0}, Imm5 基址变址寻址
sh{p}!	存储半字数据	sh! rD_{g0}, $[rA_{g0}]$ 基址寻址 shp! rD_{g0}, Imm6 基址变址寻址
sw{p}!	存储字数据	sw! rD_{g0}, $[rA_{g0}]$ 基址寻址 swp! rD_{g0}, Imm7 基址变址寻址
ldiu!	装载无符号立即数	ldiu! rD_{g0}, Imm8
push!	存储字数据（前减量变址寻址）	push! rD_{g0}, $[rA_{g0}]$
pop!	装载字数据（后增量变址寻址）	pop! rD_{g0}, $[rA_{g0}]$

[1] 立即数寻址仅用于 ldiu!指令中。格式如下：

ldiu! rD_{g0}, Imm8

立即数寻址将立即数作为操作数直接参与操作。在该指令中，是将八位立即数 Imm8 装载进低组通用寄存器 rD_{g0} 内。

指令举例如下：

```
ldiu! r3, 0x80                    // 将立即数 0x80 装载进 r3 寄存器
```

[2] 基址寻址格式如下：

<opcode>! rD_{g0}, $[rA_{g0}]$

基址寻址是以源寄存器内的内容作为操作地址对存储器进行装载/存储操作。

指令举例如下：

lh! r3, [r2]　　　　　// 将 r2 的内容为地址的存储单元的半字数据装载进 r3 寄存器

sw! r3, [r2]　　　　　// 将 r3 寄存器的字数据存储到 r2 的内容为地址的存储单元内

[3]　基址变址寻址格式如下：

<opcode>p! rD_{g0}, Imm

基址变址寻址是以基址指针寄存器 r2 的内容加上立即数偏移量做为操作地址对存储器进行装载/存储操作。在指令后加上 p 后缀则表示使用基址变址寻址。需要注意的是，在执行半字数据或字数据的装载/存储操作时，偏移量需保证地址对齐。

指令举例如下：

lhp! r3, 0x04　　　　// 将 r2 的内容加上 0x04 做为地址指向的存储单元的半字数据

　　　　　　　　　　// 装载进 r3 寄存器

swp! r3, 0x10　　　　// 将 r2 的内容加上 0x10 做为地址指向的存储单元的字数据

　　　　　　　　　　// 装载进 r3 寄存器

[4]　前减量变址寻址仅用于 push!指令中。格式如下：

push! rD_{g0}, [rA_{g0}]

前减量变址寻址以基址寄存器 rA_{g0} 的内容减去 4 做为操作地址对存储器进行操作，并且，当操作完成后，基址寄存器 rA_{g0} 的内容更新为该操作地址。push!指令完成将 rD_{g0} 寄存器的内容存储至该操作地址指向的存储单元内。

需要注意的是，存储器操作地址需为字边界对齐，否则将发生地址对齐错误异常。

指令举例如下：

push! r15, [r2]　　　　// r2 内容减 4，然后将寄存器 r15 的内容压入 r2 指向的存储单元

[5]　后增量变址寻址仅用于 pop!指令中。格式如下：

pop! rD_{g0}, [rA_{g0}]

前减量变址寻址以基址寄存器 rA_{g0} 的内容做为操作地址对存储器进行操作，并且，当操作完成后，基址寄存器 rA_{g0} 的内容更新为该操作地址加 4。pop!指令完成将操作地址指向的存储单元内的内容装载至 rD_{g0} 寄存器。

需要注意的是，存储器操作地址需为字边界对齐，否则将发生地址对齐错误异常。

指令举例如下：

pop! r15, [r2]　　　　// 将 r2 指向的存储单元的数据装载至 r15

　　　　　　　　　　// 然后将 r2 的内容加 4 并更新

2.4.2　数据处理指令

S+core 处理器的 16 位数据处理指令可以分为五类：算术运算指令、逻辑运算指令、位运算指

令、数据传送指令和移位运算指令。与 32 位指令不同的是，16 位数据处理指令均可以影响条件标志位而不需要带有.c 后缀。表 2.5 所示为 S+core 的 16 位数据处理指令表。

表 2.14 S+core16 位数据处理指令表

助记符	说明	格式	完成操作
add!	16 位加法运算	add! rD_{g0}, rA_{g0}	$rD_{g0} = rD_{g0} + rA_{g0}$
addc!	16 位带进位加法运算	addc! rD_{g0}, rA_{g0}	$rD_{g0} = rD_{g0} + rA_{g0} + C$
addei!	16 位立即数加法运算	addei! rD_{g0}, Imm4	$rD_{g0} = rD_{g0} + 2^{\wedge Imm4}$
sub!	16 位减法运算	sub! rD_{g0}, rA_{g0}	$rD_{g0} = rD_{g0} - rA_{g0}$
subei!	16 位立即数减法运算	subei! rD_{g0}, Imm4	$rD_{g0} = rD_{g0} - 2^{\wedge Imm4}$
neg!	16 位取负	neg! rD_{g0}, rA_{g0}	$rD_{g0} = -rA_{g0}$
cmp!	16 位比较	cmp! rD_{g0}, rA_{g0}	做 $rD_{g0} - rA_{g0}$ 运算并根据结果置位条件标志位
and!	16 位逻辑与	and! rD_{g0}, rA_{g0}	rD_{g0} = rD_{g0} 按位与 rA_{g0}
or!	16 位逻辑或	or! rD_{g0}, rA_{g0}	rD_{g0} = rD_{g0} 按位或 rA_{g0}
xor!	16 位逻辑异或	xor! rD_{g0}, rA_{g0}	rD_{g0} = rD_{g0} 按位异或 rA_{g0}
not!	16 位逻辑异非	not! rD_{g0}, rA_{g0}	rD_{g0} = rA_{g0} 取反
t{cond}!	16 位测试并置 T 条件标志	t{cond}!	同 32 位 t 指令
bittst!	16 位位测试	bittst! rD_{g0}, BN(Imm5)	测试 rD_{g0} 的第 BN 位并修改 N 和 Z 标志
bitset!	位置位	bitset! rD_{g0}, BN	将 rD_{g0} 第 BN 位置 1
bitclr.c	位清零	bitclr! rD_{g0}, BN	将 rD_{g0} 第 BN 位置 0
bittgl.c	位翻转	bittgl! rD_{g0}, BN	将 rD_{g0} 第 BN 位取反
mv!	16 位寄存器数据传送(低组->低组)	mv! rD_{g0}, rA_{g0}	$rD_{g0} = rA_{g0}$
mlfh!	16 位寄存器数据传送(高组->低组)	mlfh! rD_{g0}, rA_{g1}	$rD_{g0} = rA_{g1}$
mhfl!	16 位寄存器数据传送(低组->高组)	mhfl! rD_{g1}, rA_{g0}	$rD_{g1} = rA_{g0}$
sll!	16 位逻辑左移	sll! rD_{g0}, rA_{g0}	将 rD_{g0} 内容逻辑左移 rA_{g0} 位并保存至 rD_{g0}
slli!	16 位立即数逻辑左移	slli! rD_{g0}, Imm5	将 rD_{g0} 内容逻辑左移 Imm5 位并保存至 rD_{g0}

助记符	说明	格式	完成操作
sra!	16 位算术右移	sra! rD_{g0}, rA_{g0}	将 rD_{g0} 内容算术右移 rA_{g0} 位并保存至 rD_{g0}
srl!	16 位逻辑右移	srl! rD_{g0}, rA_{g0}	将 rD_{g0} 内容逻辑右移 rA_{g0} 位并保存至 rD_{g0}
srli!	16 位立即数逻辑右移	srli! rD_{g0}, Imm5	将 rD_{g0} 内容逻辑右移 Imm5 位并保存至 rD_{g0}

2.4.3 跳转与分支指令

S+core 处理器具有 j{l}!、b{cond}!和 br{cond}!三条跳转与分支指令。指令格式和功能解释如下：

j{l}!——无条件跳转

指令格式：j{l}! label

该指令首先将 label 代表的 11 位偏移地址左移 1 位，并与当前指令的高 20 位结合形成目标地址，进行分支操作。同时，若指令带有 l 后缀，则使用下一条指令的地址更新连接寄存器 GPR_{r3}。

指令举例如下：

```
j! label                // 跳转至 label 标号处，跳转偏移限制在±2K 范围内
jl! label               // 跳转至 label 标号处并保存下一条指令地址，可以用于函数调用
```

b{cond}!——条件分支

指令格式：b{cond}! label

该指令的分支条件 cond 见表 2.7。label 的有效位数为 8 位（7 位有效偏移经符号扩展并左移 1 位）。注意，该指令不具备链接功能。

指令举例如下：

```
beq! label              // Z=0 时分支至 label 标号处
```

br{cond}!——条件寄存器分支

指令格式：br{cond}! rA_{g0}

br 指令根据对条件标志位的测试结果分支到由 rA 寄存器指定的目标地址。分支条件 cond 如表 2.9 所示。

指令举例如下：

```
breq! r3                // Z=0 时分支至 r3 指定的地址处
```



2.4.4 特殊指令

S+core 处理器的 16 位指令集中仅有一条特殊控制指令：sdbbp!，用于软件 Debug 断点操作。

指令格式：sdbbp! code(Imm5)

该指令的功能与 32 位指令集中的 sdbbp 指令相同。

指令举例如下：

sdbbp! 0x00

2.4.5 并行条件执行

S+core 处理器特有并行条件执行（PCE）功能。将两条 16 位指令使用“||”连接起来，便构成了并行条件执行结构。如下所示：

ADD! r2, r7 || SUB! r2, r7

处理器根据并行执行条件标志 T 来决定执行两条指令中的哪一条。当 T=true 时，处理器执行前者，即 ADD! r2, r7；当 T=false 时，处理器执行后者，即 SUB! r2, r7。

一般并行条件执行结构的指令需要配合 T 指令共同工作。首先使用 T 指令修改 T 标志，接着使用并行条件执行结构让处理器判断应该执行哪一条指令。

并行条件执行结构有效解决了程序跳转问题，使流水线结构的处理器能更大效能的发挥其处理性能。

2.5 合成指令集

在 S+core 处理器的 32 位指令集中，部分指令可以等效为两条或两条以上的指令操作，从而方便用户的编程。

li——装载立即数

指令格式：li rD, Imm32

该指令可以将一个 32 位的立即数装载至通用寄存器 rD 内。

- 当 Imm32>-32768 且 Imm32<32767 时，该指令等效于 ldi rD, SImm16，如 li r3, 0x1234；
- 当 Imm32 的低半字为零时，该指令等效于 ldis rD, SImm16，如 li r3, 0x12340000；
- 其他情况下，该指令等效于下面两步操作：

  ldis rD, Hi16(Imm32)

  ori rD, Lo16(Imm32)

  如：li r3, 0x12345678

la——装载立即数

指令格式：　la rD, label

　　　　　　la rD, value

该指令可以将一个符号的地址装或一个 32 位的立即数装载至通用寄存器 rD 内。其中，label 为符号名，value 为 32 位立即数。

- la rD, label 等效于下面两步操作：

  ldis rD, Hi16(label)

  ori rD, Lo16(label)

  如：la r3, label

- la rD, value 等效于下面两步操作：

  ldis rD, Hi16(value)

  ori rD, Lo16(value)

  如：la r3, 0x1234

lb——装载有符号字节数据

指令格式： lb rD, [rA]

lb rD, label

lb rD, value

该指令将一个有符号字节数据经符号扩展装载进通用寄存器 rD 内。其中，label 为符号名，value 为 32 位立即数，表示存储器的操作地址（绝对寻址）。

- lb rD, [rA]等效于 lb rD, [rA, 0x0000]。

  如：lb r2, [r3]

- lb rD, label 等效于下面两步操作：

  la r1, label

  lb rD, [r1]

  如：lb r6, label

  当 label 处于 sbss 或 sdata 中，则 lb rD, label 等效于 lb rD, [gp, offset]。其中，offset 为 label 和 r28 之间的偏移量。

- lb rD, value 等效于下面两步操作：

  la r1, value

  lb rD, [r1]

  该指令使用 value 所表示的值做为地址进行寻址操作。如：lb r2, 0

lbu——装载无符号字节数据

指令格式： lbu rD, [rA]

lbu rD, label

lbu rD, value

该指令将一个无符号字节数据经零扩展装载进通用寄存器 rD 内，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

```
lbu r2, [r3]          // 等效于 lbu r2, [r3, 0]
lbu r6, label         // 等效于 la r1, label 和 lbu r6, [r1]两条指令
lbu r2, 0             // 绝对寻址，等效于 la r1, 0 和 lbu r2, [r1]两条指令
```

lh——装载有符号半字数据

指令格式： lh rD, [rA]

lh rD, label

lh rD, value

该指令将一个有符号半字数据经符号扩展装载进通用寄存器 rD 内，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

```
lh r2, [r3]          // 等效于 lh r2, [r3, 0]
lh r6, label         // 等效于 la r1, label 和 lh r6, [r1]两条指令
lh r2, 0             // 绝对寻址，等效于 la r1, 0 和 lh r2, [r1]两条指令
```

lhu——装载无符号半字数据

指令格式： lh rD, [rA]

lh rD, label

lh rD, value

该指令将一个无符号半字数据经零扩展装载进通用寄存器 rD 内，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

```
lhu r2, [r3]          // 等效于 lhu r2, [r3, 0]
lhu r6, label         // 等效于 la r1, label 和 lhu r6, [r1]两条指令
lhu r2, 0             // 绝对寻址，等效于 la r1, 0 和 lhu r2, [r1]两条指令
```

lw——装载字数据

指令格式： lw rD, [rA]

lw rD, label

lw rD, value

该指令将一个字数据装载进通用寄存器 rD 内，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

lw r2, [r3]　　　// 等效于 lw r2, [r3, 0]

lw r6, label　　　// 等效于 la r1, label 和 lw r6, [r1]两条指令

lw r2, 0　　　// 绝对寻址，等效于 la r1, 0 和 lw r2, [r1]两条指令

sb——存储字节数据

指令格式：　sb rD, [rA]

sb rD, label

sb rD, value

该指令将通用寄存器 rD 低 8 位内容写入到存储器，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

sb r2, [r3]　　// 等效于 sb r2, [r3, 0]

sb r6, label　　　// 等效于 la r1, label 和 sb r6, [r1]两条指令

sb r2, 0　　　// 绝对寻址，等效于 la r1, 0 和 sb r2, [r1]两条指令

sh——存储半字数据

指令格式：　sh rD, [rA]

sh rD, label

sh rD, value

该指令将通用寄存器 rD 低 16 位内容写入到存储器，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

sh r2, [r3]　　　// 等效于 sh r2, [r3, 0]

sh r6, label　　　// 等效于 la r1, label 和 sh r6, [r1]两条指令

sh r2, 0　　　// 绝对寻址，等效于 la r1, 0 和 sh r2, [r1]两条指令

sw——存储字数据

指令格式：　sw rD, [rA]

sw rD, label

sw rD, value

该指令将通用寄存器 rD 的内容写入到存储器，它的使用方式和等效方式与前面的 lb 类似。

指令举例如下：

sw r2, [r3]　　　// 等效于 sw r2, [r3, 0]

sw r6, label　　　// 等效于 la r1, label 和 sw r6, [r1]两条指令

sw r2, 0　　　// 绝对寻址，等效于 la r1, 0 和 sw r2, [r1]两条指令

mul/mulu——乘法运算

指令格式：　mul rD, rA, rB

　　　　　　mulu rD, rA, rB

完成 rA×rB 的操作，并将结果的低字由 CEL 传送至 rD 寄存器。操作等效于：

mul rA, rB（mulu rA, rB）

mfcel rD

指令举例如下：

mul r4, r6, r7

mulu r4, r6, r7

div/divu——除法运算

指令格式：　div rA, rB

　　　　　　divu rA, rB

完成 rA÷rB 的操作，并将商由 CEL 传送至 rD 寄存器。操作等效于：

div rA, rB（divu rA, rB）

mfceh rD

指令举例如下：

div r4, r6

divu r4, r6

rem/remu——除法运算

指令格式：　rem rD, rA, rB

　　　　　　remu rD, rA, rB

这两条指令完全等效于 div rD, rA, rB 和 divu rD, rA, rB。

2.6 S+CORE 处理器的 GNU 编译器

2.6.1 S+core7 C 编译器参数

在命令行下键入 Score-linux-elf-gcc 即可运行 S+core 处理器的 C 编译器，并可设置一些编译参

数。

语法：Score-linux-elf-gcc [option|file]

各参数内容含义如下：

-mSCORE5U	选择 S+core5U 的编译器
-mSCORE5	选择 S+core5 的编译器
-mSCORE7	选择 S+core7 的编译器
-meb/-mel	选择大端或小端模式
-S	编译成汇编语言
-E	对指定的 C 程序仅用预处理器进行预处理
-o file	将编译结果输出到文件 file 中
-help	在屏幕上列出 GCC 定义的命令行的功能描述
-ansi	支持所有 ANSI 标准 C 语言程序，并禁止那些与标准 C 语言不兼容的 GCC 程序编译特性。注意，该参数并不拒绝非标准 C 语言程序。若要 GCC 拒绝非标准 C 语言程序并产生警告信息，需将-pedantic 参数放到-ansi 后面。
-pedantic	对标准 C（ANSI C 和 ISO C++语言程序产生警告）
-w	禁止所有警告信息(warning)的输出
-Wall	允许所有警告信息(warning)的输出。这些警告是针对用户感觉可疑的问题，且这些问题很容易避免或修正，甚至与宏相关的问题亦是如此
-Werror	将所有警告性质的问题设置成错误(error)性质
-Q	编译时显示被编译的函数名称列表，并同时显示编译过程中各阶段的耗时信息
-D*macro*	定义名称为 macro 的宏
-D*macro=defn*	定义名称为 macro 的宏的值为 defn。命令行中的所有“-D”参数都在处理“-U”参数前被处理
-U*macro*	取消定义名称为 macro 的宏
-gstab+	为调试器(debuggers)产生调试(debug)信息
-I*dir*	在头文件里的搜索目录列表中增添名为“dir”的目录
-O0	不进行优化处理
-O1	编译器进行关于减少代码容量及执行时间的优化处理
-O2	比 O1 更进一步优化，执行除了容量-速度折中优化之外的所有优化措施
-O3	比 O2 更进一步优化，除了 O2 所有优化措施外，还增加内联(in-lining)功能
-Os	优化容量，即允许所有 O2 优化，又不明显增加代码容量，且做更进一步减少代码容量的优化
-nostartfiles	不链接 crt*.o

-nostdlib　　不链接标准库(libc.a, libm.a……)

举例：

例 1：

在命令行窗口中键入：gcc -S test.c -o test.s

其中，test.c 是一个 C 语言代码文件的名称；该命令会产生一名为 test.s 的汇编语言代码文件。

例 2：

在命令行窗口中键入：gcc -S -gstabs+ test.c -o test.s

其中，test.c 是一个 C 语言代码文件的名称；该命令会产生一名为 test.s 的汇编语言代码文件，并且包含有 Debug 信息。

例 3：

在命令行窗口中键入：gcc -S -O2 –gstabs+ test.c -o test.s

其中，test.c 是一个 C 语言代码文件的名称；该命令会产生一名为 test.s 的经过 O2 优化的汇编语言代码文件，并且包含有 Debug 信息。

2.6.2 S+core7 C 编译器的基本数据类型

S+core 处理器的编译器支持表 2.15 所示的数据类型。

表 2.15 S+core7 C 编译器的基本数据类型

数据类型	位数
char/unsigned char	8
short/unsigned short	16
int/unsigned int	32
long/unsigned long	32
long long/unsigned long long	64
float	32 位 IEEE 浮点格式
double	64 位 IEEE 浮点格式

2.6.3 S+core7 C 编译器的函数调用约定

用户可以在 C 语言中直接使用汇编中的标号代表函数名来调用汇编函数，而无需任何其他转换。同样，用户可以在汇编语言中直接使用形如“bl <C 函数名>”这样的指令直接调用使用 C 语言编写的函数。

S+core7 处理器有 32 个通用寄存器。这里，将分类说明这些寄存器的用法，具体见表 2.16。

表 2.16 S+core7 通用寄存器在编译器中的使用分配

寄存器	功能描述	函数调用时是否被保存

寄存器	功能描述	函数调用时是否被保存
r0	堆栈指针	是
r1	暂存器，通常用于汇编器	否
r2	帧（页）指针	是
r3	链接寄存器。函数调用时，此寄存器保存返回地址	是
r4~r7	用于传递参数及返回值	否
r8~r11	调用者保存的寄存器	否
r12~r21	被调用者保存的寄存器，或选择用于保存帧指针	是
r22~r27	调用者保存的寄存器	否
r28	全局指针	是
r29	编译器保留	是
r30~r31	仅为操作系统使用	是
hi	乘/除法运算特殊寄存器，保存乘法运算结果的高 32 位或除法结果的余数	否
lo	乘/除法运算特殊寄存器，保存乘法运算结果的低 32 位或除法结果的商	否
sr0	特殊寄存器用于计数器操作	否
sr1	特殊寄存器用于装载合并指令操作	否
sr2	特殊寄存器用于存储合并指令操作	否

C 编译器用寄存器 r4~r7 传递第 1 至第 4 个参数，且将返回值置于寄存器 r4 中。如果有多于 4 个参数的函数，则第 5 个参数会被置于存储器[sp+16]中，而第 6 个参数会被置于存储器[sp+20]中，以此类推。用户在处理汇编与 C 相互调用期间的参出及返回值传递时需要注意。

用法举例：

这里列举一个参数传递的例子，上面的程序为 C 语言源程序，下面的程序为编译器编译出来的汇编语言程序，由此可以看到，第 5 参数被置于[sp+16]存储器里，而第 6 参数被置于[sp+20]存储器里。

```
void arg6(int a, int b, int c, int d, int e, int f);
int main(void)
{
    arg6(1,2,3,4,5,6);
```

```
}

void arg6(a, b, c, d, e, f)
        int a, b, c, d, e, f
{
}
```

对应的汇编代码如下：

```
.text
        .align 2
        .globl main
main:
        sw r2, [r0, -4]+
        sw r3, [r0, -4]+
        sw r0, [r0, -4]+
        subi r0, 24
        mv r2, r0
        la r8, __main
        brl r8
        li r8, 5
        sw r8, [r0, 16]
        li r8, 6
        sw r8, [r0, 20]
        li r4, 1
        li r5, 2
        li r6, 3
        li r7, 4
        la r8, arg6
        brl r8
        addi r2, 24
        lw r0, [r2]+, 4
        lw r3, [r0]+, 4
        lw r2, [r0]+, 4
```

```
        br r3

        .align 2
        .globl arg6
arg6:
        sw r2, [r0, -4]+
        sw r0, [r0, -4]+
        mv r2, r0
        lw r0, [r2]+, 4
        lw r2, [r0]+, 4
        br r3
```

2.7 S+CORE 处理器的 GNU 汇编器

2.7.1 S+core7 C 汇编器参数

在命令行下键入 Score-linux-elf-as 即可运行 S+core 处理器的 C 编译器，并可设置一些编译参数。

语法：Score-linux-elf- as [option|file]

各参数内容含义如下：

-SCORE5U　指示汇编器按照 SCORE5U 产生汇编

-SCORE5　指示汇编器按照 SCORE5 产生汇编

-SCORE7　指示汇编器按照 SCORE7 产生汇编

-EB　选择大端模式（默认参数）

-EL　选择小端模式

-FIXDD　指示汇编器解决数据依赖问题（当前不支持）

-NWARN　汇编器不产生关于数据依赖问题的警告信息

-USE_R1　汇编器不产生关于 R1 寄存器（汇编器用暂存器）的警告信息

-G0　Gp 开关控制值为 0，即不使用 gp（全局指针）寻址方式

-gstabs　对每一汇编指令行都产生 stabs 格式的调试信息

-I *dir*　在头文件的搜索目录列表中增添名为 dir 的目录，该目录中的文件用伪指令.include 包含进源文件中

-o *objfile*　指定汇编器的输出文件名为“objfile”

-O0　汇编器不对汇编代码进行优化

举例：

在命令行窗口中键入：Score-linux-elf-as –SCORE7 -EL -USE_R1 -gstabs -I . -o test.o test.s

其中，test.s 文件是一个汇编程序文件，该命令会产生一名为 test.o 的针对 SCORE7 的小端格式存储的不包含关于 R1 寄存器的目标文件。

2.7.2 汇编语言语法

汇编语言源程序包含一系列的汇编语句，一行一句。每条语句都有如下的格式，且每一部分都可根据情况选择是要还是不要。

Label: instruction #comment

其中，Label（标号）是允许编程者以标号的形式指定程序中某一地址，即 PC 的位置。一个标号可以用任意一有效符号后面加上“:”来表示。所谓有效符号是指由字母字符 A~Z、a~z、数字字符 0~9 以及字符“_”、“-”和“$”组成的字符串，并且字符串不能以数字开头。

comment（注释）必须跟在符号“#”的后面，亦即任何跟在符号“#”后面的字符（除#include 和#define 外）都会被汇编器忽略掉。在此，也可以用 C 语言程序的注释符（/*和*/或//）来引用注释句。

instruction（指令）是用户程序的实质内容，用户既可以使用 S+core 处理器的任何汇编语言指令（用于汇编成 S+core 处理器要执行的操作码），也可以使用伪指令，用于汇编器要执行的伪指令操作。伪指令的作用及其用法将在后面讨论。

2.7.3 汇编器伪指令

所有的伪指令名前都必须用符号“.”开头。这里，以字母顺序列出用户汇编程序中可能会经常用到的伪指令。

```
.align
    语法：
        .align alignment[, [fill][, max]]
    用法：
        .align 1            // 将位置计数器 PC 对齐到值为 2 的 1 次幂的地址（半字对齐）
                            // 若 PC 的值已经是 2 的 1 次幂整数倍，则不需要任何改变
        .align 2            // 将位置计数器 PC 对齐到值为 2 的 2 次幂的地址（字对齐）
                            // 若 PC 的值已经是 2 的 2 次幂整数倍，则不需要任何改变
.ascii
    语法：
        .ascii strings
    用法：
        .ascii “JNZ”        // 插入字符串“JNZ”，即插入字节数据 0x4a 0x4e 0x5a
.asciz
```

```
    语法：
        .asciz strings
    用法：
        .asciz “JNZ”            // 插入包含有字符串结束标志的字符串”JNZ”，
                                // 即插入字节数据 0x4a 0x4e 0x5a 0x00
.byte
    语法：
        .byte expressions
    用法：
        .byte 64, ‘A’           // 插入字节数据 0x40 0x41
        .byte 0x42              // 插入字节数据 0x42（0x 或 0X 均为 16 进制数的前缀）
        .byte 0b1000011, 0104   // 插入字节数据 0x43, 0x44
                                // 0b 或 0B 均为 2 进制数的前缀
                                // 0 为 8 进制数的前缀
.data
    语法：
        .data [subsection]
    用法：
        .data                   // 切换到.data 段
.equ
    语法：
        .equ symbol, expression
    用法：
        .equ zzz, (5*8)+2       // 全局符号 zzz 的地址赋值为 42
                                // 本伪指令等效于操作：zzz = 42
.extern
    语法：
        .extern symbol
    用法：
        .extern label_zzz       // 指明符号 label_zzz 定义在其他源文件里
.global
    语法：
```

```
        .global symbol
    用法：
        .global _start          // 指明_start 为所有模块使用的全局符号，包括链接器
.hword
    语法：
        .hword expression
    用法：
        .hword 0xaa55, 12345    // 插入半字数据，存储顺序为 0x55, 0xaa, 0x39, 0x30
.include
    语法：
        .include filename
    用法：
        .include “aaa”          // 引入其他文件到当前文件内
.int
    语法：
        .int expressions
    用法：
        .int aaa                // 在未初始化的段里插入符号 aaa 的 4 个字节数据
.org
    语法：
        .org new-lc [, fill]
    用法：
        .org 0x1234             // 将当前段的位置计数器 PC 后移到地址 0x1234
.section
    语法：
        .section NAME [, "FLAGS"[, @TYPE[, @ENTSIZE]]]
        FLAGS 常用的有以下几种：
        `a'      可分配段
        `w'      可写段
        `x'      可执行段
        data section 具有的 FLAGS 通常是’wa’；
        text section 具有的 FLAGS 通常是’ax’。
```

```
    用法：
        .section .text1, “wa”     // 定义一个.text1 段，且该段为可写的以及可分配的
.set
    语法：
        .set symbol, expression
    用法：
        .set nor1                 // 从当前位置开始，若使用 r1，则汇编器输出警告信息
        .set r1                   // 从当前位置开始，若使用 r1，则汇编器不会输出警告信息
        .set volatile             // 汇编器不对用户代码进行优化
        .set optimize             // 汇编器会根据代码大小对用户代码进行优化
        .set nwarn                // 当发生数据依赖性问题时，汇编器不会输出警告信息
.space
    语法：
        .space size [, fill]
    用法：
        .space 100                // 从当前位置插入 100 个字节数据 0x00
.text
    语法：
        .text
    用法：
        .text                     // 切换到.text 段
.word
    语法：
        .word expression
    用法：
        .word 0xdeadbeaf          // 插入字数据，存储顺序为 0xaf, 0xbe, 0xad, 0xde
```

2.7.4 段及其重定位

简单地说，“段”是指一个连续的地址空间，所有处于该地址范围内的数据均会被处理为具有相同性质的一批数据，譬如“只读”段。

链接器 ld 将从各个目标文件(程序的一部分)读出的内容链接到一起，形成一个可运行程序。汇编器 as 输出的每一个目标文件，其程序的起始地址都会被假定为 0；而链接器 ld 会为其安排一个最终的运行地址，以保证所有这些目标文件中的程序地址不会重叠。这实际是一种简化操作，但其

足以解释汇编时“段”的作用。

链接器 ld 将用户程序中的字节块整体移到运行地址(run-time address)上，即字节块长度以及块内字节顺序均不会改变。这个块整体单元就被称为“段”，而为段安排运行地址的过程就称为“重定位”，即将目标文件地址调整到正确的运行地址上。

汇编器 as 产生的目标文件至少包含 3 个段：.text 段、.data 段 和.bss 段。其中，每一段既可以有数据，也可以没有数据。

除此之外，用户还可以通过伪指令.section 来定义一个段。假如用户未使用伪指令来定义.text 段或.data 段，而这些段依然存在，只是里面没有数据。通常，目标文件的.text 段起始在地址 0 处，.data 段则跟在其后，而.bss 段则跟在.data 段之后。

为了链接器 ld 能执行重定位，汇编器 as 需要把有关重定位的一些详细信息写到目标文件中。这些信息主要包括：

- 目标文件中该引用的起始地址在哪里？
- 该引用的长度为多少字节？
- 该引用的定义处于另一个目标文件中的哪一段？相对于那个段的起始地址的偏移量是多少？
- 该引用是否为 PC 相对地址？

实际上，汇编器汇编的每一个地址都可以表示为：地址＝段起始地址＋段内偏移量

由于多数用于计算地址的表达式都具有这种“段－段内相对地址”的属性，因此，可以使用符号{secname N}来表示段名为 secname 的段内的偏移量 N 这样一个意思。

除了上述.text、.data 和.bss 段之外，还需要了解关于绝对段(absolute section)的概念。当链接器 ld 在链接各目标文件时将部分程序的地址重定位后，处于绝对段的这些地址就不会改变了。例如，ld 会为绝对地址{absolute 0}重定位到值为 0 的运行地址。尽管 ld 不会为两个目标文件程序的数据段分配重叠的地址，但其由绝对段定义的程序地址却很可能会重叠，如其中一部分程序的地址（绝对地址为 239）可能总会与另一部分程序要运行的地址（其绝对地址亦为 239）相同。

段的概念可以延伸到未定义段(undefined section)。任何地址所处的段在汇编时都是未知的，通过符号{undefined U}标记出，其中 U 表示在链接时要被填充的意思。由于数字总是表示已经定义过的，产生未定义地址的唯一方法就是引用一个未定义符号。common 类型的变量就属于未定义符号：该符号的值在汇编阶段是未知的，因此它位于未定义段。

通过类推可知，用“section”这个字来表示链接程序中相同性质的段的集合。链接器 ld 在进行程序链接时惯常的做法是将所有目标文件中程序的.text 段连续摆放，集合成一个“大”.text 段。同样，对.data 段和.bss 段也有类似的处理。

2.8 S+CORE 处理器的 GNU 链接器

S+core 处理器的链接器的命令行参数如下：

-l*archive* 添加文件名为 *archive* 的档案文件到要链接的文件列表中

-L*searchdir*　将名为 *searchdir* 的路径添加到路径列表中，该路径列表为链接器 ld 搜索各文件库用

-M　将链接映射文件列印在链接器标准输出中

-o *output*　指定文件名为 *output* 的文件为链接器 ld 链接操作后的输出文件

-EB　链接大端格式的目标（默认参数）

-EL　链接小端格式的目标

T*scriptfile*　从名为 *scriptfile* 的文件里读出链接命令，这些链接命令会取代链接器 ld 默认的链接参数

举例：

在命令行窗口中键入：Score-linux-elf-ld -Texec.ld test.o -L . -o test.exec

其中，test.o 文件是一个目标文件，该命令会根据 exec.ld 链接脚本文件的内容将 test.o 文件链接并产生一个名为 test.exec 的可执行文件。

3 SPCE3200 使用指南

3.1 简介

3.1.1 概述

SPCE3200 是一款高度集成的为多媒体应用设计的高性能 32 位芯片。它采用凌阳科技独立知识产权的 32 位 Score 处理器为内核，内置 MPEG4 硬件编解码模块，并外扩其它用于多媒体、机器人领域的功能模块。它专长于图像、视频处理，可以输出图像、声音到电视机(NTSC 或 PAL 制式)以及 LCD 上显示，具备强大的音频、视频、图像数据的处理能力。它可以输出丰富的视频画面、声音，并将这些数据存储到 SD 卡或 NAND Flash 上。

SPCE3200 的工作电压范围为 3.0V~3.6V，CPU 频率为 27~162MHz。此外，芯片提供 32768Hz 实时时钟、低电压检测、低电压复位、12 位模数转换器(ADC)、UART 接口、SPI 接口、SIO 接口、I2C 主设备接口以及其它 I/O 设备接口，例如 TFT LCD、彩色 STN LCD、CMOS 图像传感器(CMOS image sensor)、TVE 控制器、光笔、触摸屏等。

S+core 7 是一个单任务的、具有 7 级流水线的高性能、高速的 32 位 RISC 处理器，采用了 Sunplus ISA（Instruction Set Architecture）指令集，支持 32 位与 16 位混合指令模式以及并行条件执行，从而提高了代码密度。在 SPCE3200 芯片中，S+core 7 运行速度可达 162MHz，为实现 Soc 集成采用了 AMBA 总线，并设计了协处理器以及 Custom Engine 接口从而可以提供灵活的扩展功能接口，并为高效地调试及在线仿真(ICE)程序采用了 SJTAG 模块。该处理器支持 4KB 的 2 路组相连的 I/D Cache，以及 4KB 的 LIM/LDM(Local Instruction/Data Memory)。S+core 7 系列 CPU 支持最多 63 个优先级的中断，可以快速响应中断事件。此外，提供了一些高性能的指令来实现特定功能，如：SUNPLUS 已申请专利的非对齐数据装载/存储指令以及前/后增量寻址指令，用于实现字串拷贝或存储器数据传输；位操作指令和分支指令采用循环控制计数器，用于有效地控制程序的走向；Sleep 指令为省电系统提供良好的支持。

3.1.2 SPCE3200 特性

- 工作电压：I/O 端口的 VDD 为 3.0V～3.6V，CPU 内核的 VDD 为 1.62V～1.98V
- CPU 工作频率：27～162 MHz
- 支持扩展 SDR DRAM 和 DDR DRAM，最大容量可达 16M 字节
- 具有 32 位/16 位的 SDRAM 数据总线
- 支持隔行扫描/逐行扫描的 NTSC/PAL 视频输出
- 图像分辨率：VGA 模式为(640 像素 x 480 像素)；CIF 模式为(320 像素 x 240 像素)
- 支持 65536 色(RGB565 格式)
- 可编程选择颜色模式：4/16/64/256/32768/65536
- 硬件方式的 MPEG-4/JPEG 编解码
- MPEG-4 帧率(frame rate)：CIF 模式下高达 30 帧/秒

- 4 通道 APB DMA 数据传输方式：从 APB 设备到 DRAM，或从 DRAM 到 APB 设备
- 硬件的 DRAM DMA 数据传输方式：由硬件执行的 DRAM 到 DRAM 的数据传输
- 双通道 16 位高速 DAC，确保立体声输出质量
- 内置 3 个可编程锁相环(PLL)电路，为系统提供各路时钟
- 为 NTSC 制/PAL 制系统提供 27 MHz 晶振
- 具备实时时钟(RTC)
- 共 6 个 16 位定时器/计数器(具可编程自动重载功能)
- 提供 40 个中断源：分别为定时器、时基、外部输入以及键唤醒类型
- 支持键唤醒功能
- 9 通道 12 位 ADC
- USB 功能：支持 USB1.1 主机或 USB1.1 外设
- UART 功能：具有通用异步接收机和发送机
- 提供串行外围设备接口(SPI)：具主/从模式
- 提供 Sunplus 同步串行输入/输出接口（SIO）
- 内置 Watchdog 功能
- 提供 LCD 接口：具 TFT 方式/CSTN 方式
- 支持 CCIR-601/656 CMOS 影像传感器/TVE 控制接口
- 支持 SD 卡和 NAND 型 FLASH，用于海量数据存储

3.2 引脚信息

3.2.1 SPCE3200 的引脚分布

SPCE3200 共有 256 个引脚，封装形式为 PLCC256，它的引脚如图 3.1 所示。

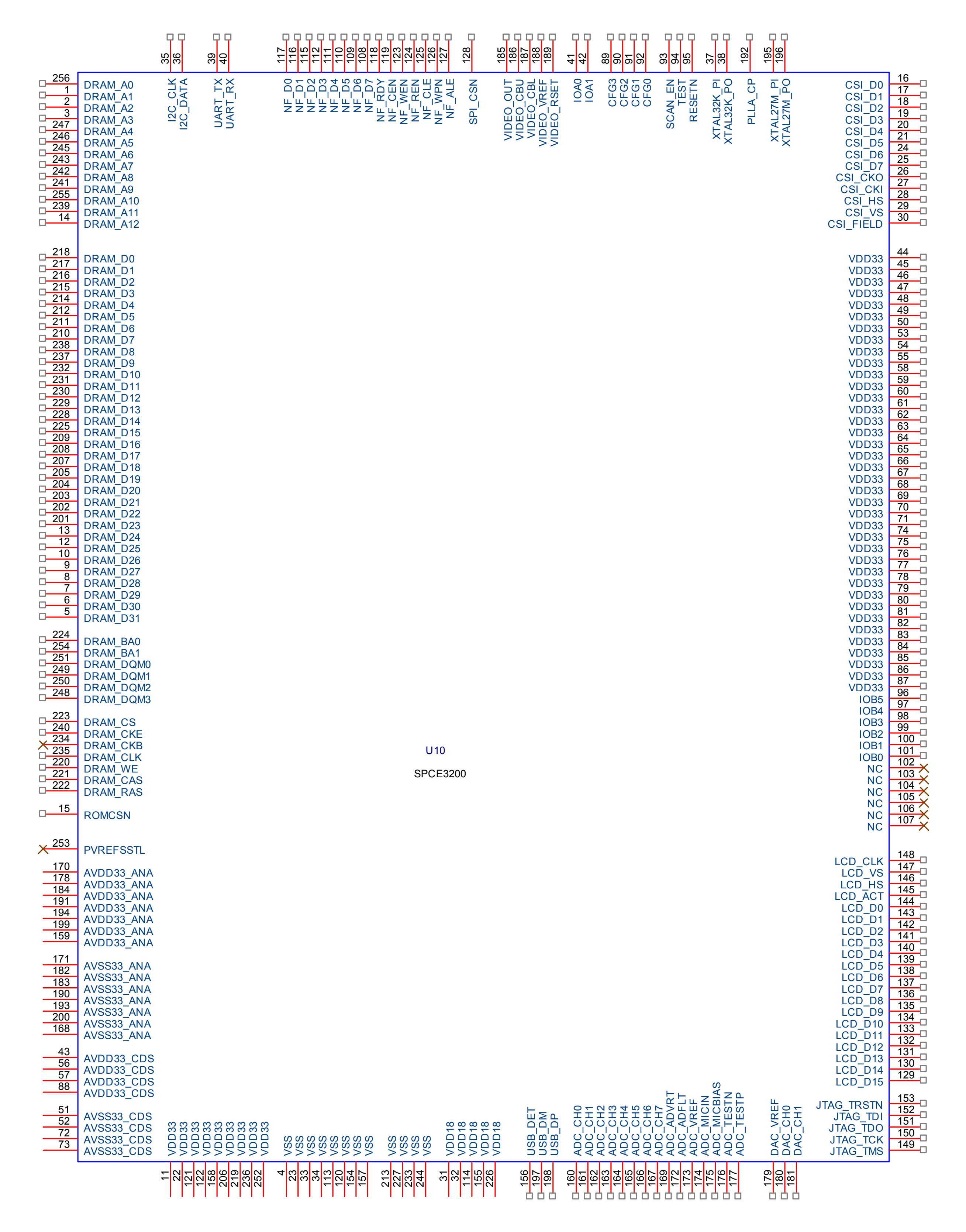


图 3.1 SPCE3200 芯片引脚图

3.2.2 SPCE3200 的引脚描述

SPCE3200 引脚描述如表 3.1 所示。

表 3.1 SPCE3200 引脚描述

引脚名称	引脚号	引脚属性	引脚功能
DRAM_A0	256	O	DRAM 地址总线
DRAM_A1	1		
DRAM_A2	2		
DRAM_A3	3		
DRAM_A4	247		
DRAM_A5	246		
DRAM_A6	245		
DRAM_A7	243		
DRAM_A8	242		
DRAM_A9	241		
DRAM_A10	255		
DRAM_A11	239		
DRAM_A12	14		
DRAM_D0	218	I/O	DRAM 数据总线
DRAM_D1	217		
DRAM_D2	216		
DRAM_D3	215		
DRAM_D4	214		
DRAM_D5	212		
DRAM_D6	211		
DRAM_D7	210		
DRAM_D8	238		
DRAM_D9	237		
DRAM_D10	232		
DRAM_D11	231		

引脚名称	引脚号	引脚属性	引脚功能
DRAM_D12	230		
DRAM_D13	229		
DRAM_D14	228		
DRAM_D15	225		
DRAM_D16	209		
DRAM_D17	208		
DRAM_D18	207		
DRAM_D19	205		
DRAM_D20	204		
DRAM_D21	203		
DRAM_D22	202		
DRAM_D23	201		
DRAM_D24	13		
DRAM_D25	12		
DRAM_D26	10		
DRAM_D27	9		
DRAM_D28	8		
DRAM_D29	7		
DRAM_D30	6		
DRAM_D31	5		
ROMCSN	15	O	Flash 片选。用于外接 Nor-type Flash 或 SRAM
CSI_D0	16	I	CMOS 传感器接口数据总线
CSI_D1	17		
CSI_D2	18		
CSI_D3	19		
CSI_D4	20		

引脚名称	引脚号	引脚属性	引脚功能
CSI_D5	21		
CSI_D6	24		
CSI_D7	25		
CSI_CKO	26	O	CMOS 传感器接口时钟输出
CSI_CKI	27	I	CMOS 传感器接口时钟输入
CSI_HS	28	I/O	CMOS 传感器接口水平同步信号
CSI_VS	29	I/O	CMOS 传感器接口垂直同步信号
CSI_FIELD	30	I/O	CMOS 传感器接口场扫描信号
I2C_CLK	35	O	I2C 时钟
I2C_DATA	36	I/O	I2C 数据
XTAL32K_PI	37	I	32768Hz 时钟输入
XTAL32K_PO	38	O	32768Hz 时钟输出
UART_TX	39	O	UART 发送
UART_RX	40	I	UART 接收
IOA0	41	I/O	A 组通用输入输出口
IOA1	42	I/O	A 组通用输入输出口
VDD33	44~87	I	输入输出口电压输入
CFG0	92	I	系统配置端口
CFG1	91		
CFG2	90		
CFG3	89		
SCAN_EN	93	I	扫描使能
TEST	94	I	测试用，接高
RESETN	95	I	复位，低电平有效
IOB5	96	I/O	B 组通用输入输出口
IOB4	97	I/O	B 组通用输入输出口

引脚名称	引脚号	引脚属性	引脚功能
IOB3	98	I/O	B 组通用输入输出口
IOB2	99	I/O	B 组通用输入输出口
IOB1	100	I/O	B 组通用输入输出口
IOB0	101	I/O	B 组通用输入输出口
NC	102~107	O	空脚
NF_D0	117	I/O	NANDFlash 数据总线
NF_D1	116		
NF_D2	115		
NF_D3	112		
NF_D4	111		
NF_D5	110		
NF_D6	109		
NF_D7	108		
NF_RDY	118	I	NANDFlash Ready 信号
NF_CEN	119	O	NANDFlash 芯片使能
NF_WEN	123	O	NANDFlash 写使能
NF_REN	124	O	NANDFlash 读使能
NF_CLE	125	O	NANDFlash 命令锁存使能
NF_WPN	126	O	NANDFlash 写保护
NF_ALE	127	O	NANDFlash 地址锁存
SPI_CSN	128	I/O	SPI 芯片使能
LCD_D0	144	I/O	LCD 接口数据总线
LCD_D1	143		
LCD_D2	142		
LCD_D3	141		
LCD_D4	140		

引脚名称	引脚号	引脚属性	引脚功能
LCD_D5	139		
LCD_D6	138		
LCD_D7	137		
LCD_D8	136		
LCD_D9	135		
LCD_D10	134		
LCD_D11	133		
LCD_D12	132		
LCD_D13	131		
LCD_D14	130		
LCD_D15	129		
LCD_ACT	145	O	LCD 接口使能信号
LCD_HS	146	O	LCD 接口水平同步信号
LCD_VS	147	O	LCD 接口垂直同步信号
LCD_CLK	148	O	LCD 接口时钟信号
JTAG_TMS	149	I	JTAG 模式选择信号
JTAG_TCK	150	I	JTAG 时钟信号
JTAG_TDO	151	O	JTAG 数据输出信号
JTAG_TDI	152	I	JTAG 数据输入信号
JTAG_TRSTN	153	I	JTAG 复位，低电平有效
USBDET	156	I	USB 侦测信号
ADC_CH0	160	I/O	ADC 模拟输入通道
ADC_CH1	161		
ADC_CH2	162		
ADC_CH3	163		
ADC_CH4	164		

引脚名称	引脚号	引脚属性	引脚功能
ADC_CH5	165		
ADC_CH6	166		
ADC_CH7	167		
ADC_ADVRT	169	I	ADC 参考电压
ADC_ADFLT	172	O	反锯齿波输出脚
ADC_VREF	173	O	AFE 参考电压
ADC_MICIN	174	I	MIC 输入口
ADC_MICBIAS	175	O	缓存适配电压为驻极体 MIC 提供偏压。电压为 AVDD33 的 3/4
ADC_TESTN	176	I/O	测试模式的负输入或输出
ADC_TESTP	177	I/O	测试模式的正输入或输出
DAC_VREF	179	I	DAC 参考电压
DAC_CH0	180	O	DAC 通道 0
DAC_CH1	181	O	DAC 通道 1
VIDEO_OUT	185	O	DAC 电流输出
VIDEO_CBU	186	O	接 0.1uF 到 AVDD
VIDEO_CBL	187	O	接 0.1uF 到 AVDD
VIDEO_VREF	188	O	能带输出
VIDEO_RSET	189	O	接 1.4K 到 GND
PLLA_CP	192	I	滤波回路
XTAL27M_PI	195	I	27MHz 晶振输入脚
XTAL27M_PO	196	O	27MHz 晶振输出脚
USB_DM	197	I/O	USB 数据负
USB_DP	198	I/O	USB 数据正
DRAM_WE	220	O	DRAM 写使能
DRAM_CAS	221	O	DRAM 柱地址开关

引脚名称	引脚号	引脚属性	引脚功能
DRAM_RAS	222	O	DRAM 行地址开关
DRAM_CS	223	O	DRAM 片选
DRAM_BA0	224	O	DRAM Bank 地址
DRAM_BA1	254		
DRAM_CKB	234	O	DRAM 时钟
DRAM_CLK	235	O	DRAM 时钟
DRAM_CKE	240	O	DRAM 时钟使能
DRAM_DQM0	251	O	DRAM 数据屏蔽，就是将 DRAM 数据输入或数据输出屏蔽起来不输入或输出
DRAM_DQM1	249		
DRAM_DQM2	250		
DRAM_DQM3	248		
PVREFSSTL	253	O	SSTL2 参考电压
AVDD33_ANA	159,170,178,184,191,194,199	I	模拟电压电源
AVSS33_ANA	168,171,182,183,190,193,200	I	模拟电压地
VDD33	11,22,121,122,158,206,219,236,252	I	I/O 电压电源
VSS	4,23,33,34,113,120,	I	I/O 电压地

引脚名称	引脚号	引脚属性	引脚功能
	154,157, 213,227, 233,244		
VDD18	31,32, 114,155, 226	I	逻辑电压电源

3.3 结构概述

SPCE3200 的结构如图 3.2 所示。它包含一个支持使用 SJTAG 进行在线仿真的 S+core 处理器 CPU、符合 AMBA 标准的与片内高效能模块相连的 AHB 总线、连接片内其他低速外设的 APB 总线以及 AHB 到 APB 总线的桥控制器和 DMA 控制器。

下图中，LDM（Local Data Memory）指片内的 8KB 的 RAM。

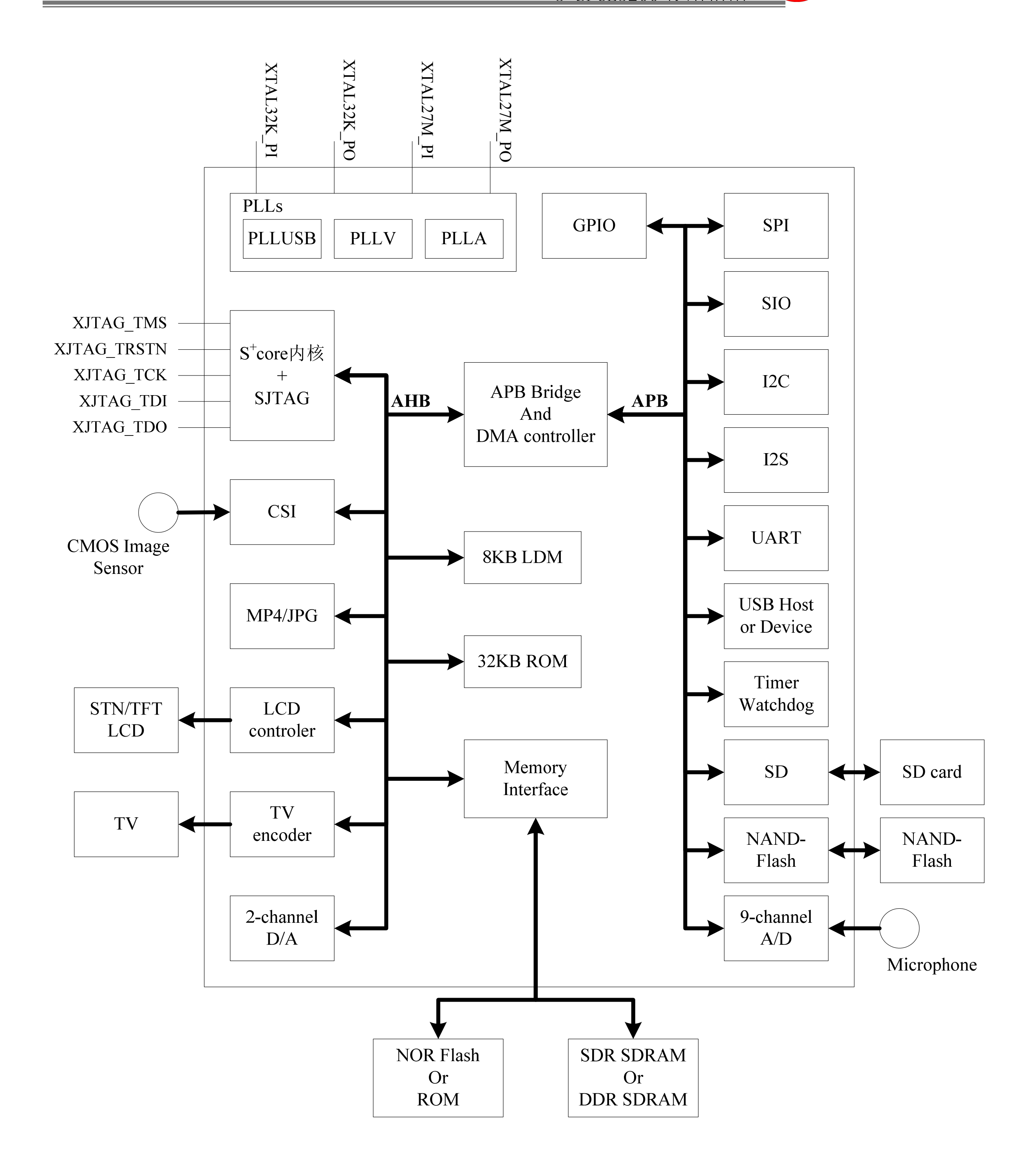


图 3.2 SPCE3200 结构框图

其中，挂载在高性能 AHB 总线上的所有外设模块包括：

- 存储器接口控制单元（MIU）
- MPEG4/JPEG 编解码模块
- CMOS 图象传感器接口（CSI）模块
- LCD 控制器模块

- TV 编码模块
- 2 通道 16 位高速 D/A 转换器

挂载在 APB 外设总线上的所有模块包括：

- GPIO 控制器模块
- 定时器、实时时钟和时间基准信号发生模块
- 看门狗模块
- SPI 串行总线控制器
- SIO 串行总线控制器
- I2C 串行总线控制器
- I2S 主/从控制器
- UART 控制器
- USB 主/从控制器
- SD 卡控制器
- Nand 型 Flash 控制器
- 9 通道 12 位 A/D 转换器

3.4 存储器映射

SPCE3200 采用基于哈佛结构的 S+core 处理器内核，处理器内部总线实行数据和指令独立编址，CPU 通过分别的总线实现对数据和程序的访问。需要注意的是，虽然处理器内核采用独立的数据总线和指令总线，但处理器采用固定存储器映射（Fixed-MMU）模式对应于外围模块和外部物理存储器，从而变为外设的统一编址。在芯片上电之后，提供给用户经过如图 3.3 所示的存储器映射之后的虚拟地址空间。

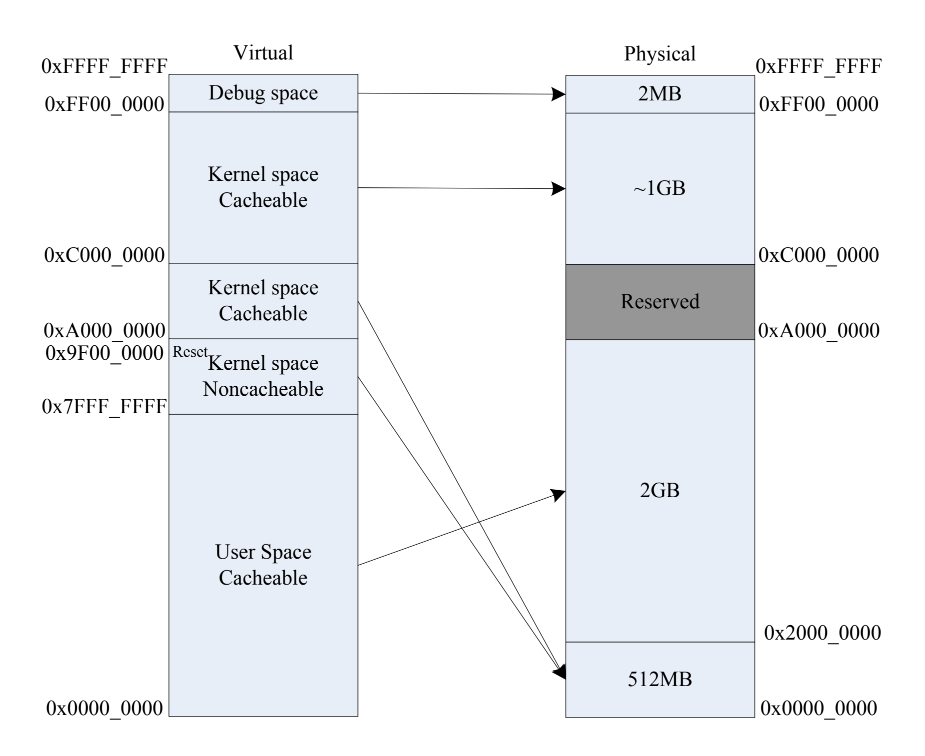


图 3.3 SPCE3200 的存储器映射示意图

SPCE3200 为每个片内外设分配了 32K 的空间，位于虚拟地址 0x88000000 开始的地址空间。这些 32K 的空间包含了各个外设的控制寄存器和数据寄存器等，用户可以直接访问这些地址来控制各个外设的工作。每个外设分配的地址空间如表 3.2 所示。

表 3.2 SPCE3200 外设地址空间分配表

地址空间	外设名称	地址空间	外设名称
0x88000000~0x8800FFFF	CSI	0x88161000~0x88161FFF	TIMER1
0x88030000~0x8803FFFF	TV 编码器	0x88162000~0x88162FFF	TIMER2
0x88040000~0x8804FFFF	LCD 驱动器	0x88163000~0x88163FFF	TIMER3
0x88070000~0x8807FFFF	MIU	0x88164000~0x88164FFF	TIMER4
0x88080000~0x8808FFFF	APBDMA	0x88165000~0x88165FFF	TIMER5
0x88090000~0x8809FFFF	缓冲区控制器	0x88166000~0x88166FFF	RTC
0x880A0000~0x880AFFFF	IRQ 控制器	0x88170000~0x8817FFFF	看门狗
0x880C0000~0x880CFFFF	LDMDMA	0x88180000~0x8818FFFF	SD 卡控制器
0x880D0000~0x880DFFFF	BLNDMA	0x88190000~0x8819FFFF	Nand 型 Flash 控制器
0x880F0000~0x880FFFFF	AHB 解码器	0x881A0000~0x881AFFFF	A/D 转换器
0x88100000~0x8810FFFF	GPIO 控制器	0x881B0000~0x881BFFFF	USB Device 控制器

地址空间	外设名称	地址空间	外设名称
0x88110000~0x8811FFFF	SPI 控制器	0x881C0000~0x881CFFFF	USB Host 控制器
0x88120000~0x8812FFFF	SIO 控制器	0x88200000~0x8820FFFF	软件配置控制器
0x88130000~0x8813FFFF	I2C 控制器	0x88210000~0x8821FFFF	时钟控制器
0x88140000~0x8814FFFF	I2S 控制器	0x88220000~0x8822FFFF	MPEG4 编解码控制器
0x88150000~0x8815FFFF	UART	0x88230000~0x8823FFFF	MIU2
0x88160000~0x88160FFF	TIMER0	0x88240000~0x8824FFFF	ECC 校验控制器

在 SPCE3200 内部还具有 8KB 的 RAM（Local Data Memory）和 32KB 的 Flash ROM（Local Instruction Memory）。在系统复位后，内部 RAM 被定位在 0xA0000000 开始的 8KB 地址空间内，内部 Flash ROM 被定位在 0x9F000000 开始的 32KB 地址空间内，程序将从内部的 Flash ROM 开始执行。同时，用户可以使用 cache 指令重新为内部 RAM 和内部 ROM 指定映射地址空间，详细请参考指令系统一章中关于 cache 指令的描述。

3.5 锁相环 PLL 与时钟发生器 CKG

3.5.1 锁相环 PLL

SPCE3200 依靠外部的一颗 27MHz 的晶振和一颗 32768Hz 的实时时钟晶振维持整个系统的运转。其中，27MHz 晶振可以直接为 APB 总线及其上的 APB 外设等模块提供工作频率，32768Hz 的实时时钟晶振可以为 Watch-Dog、Timer、RTC 和 TimeBase 等模块直接提供工作频率。

同时，SPCE3200 还内嵌三个 PLL 电路，分别称之为 PLLU、PLLV 和 PLLA。三个 PLL 电路可以将 27MHz 的晶振频率倍频到不同的频率。如图 3.4 所示。

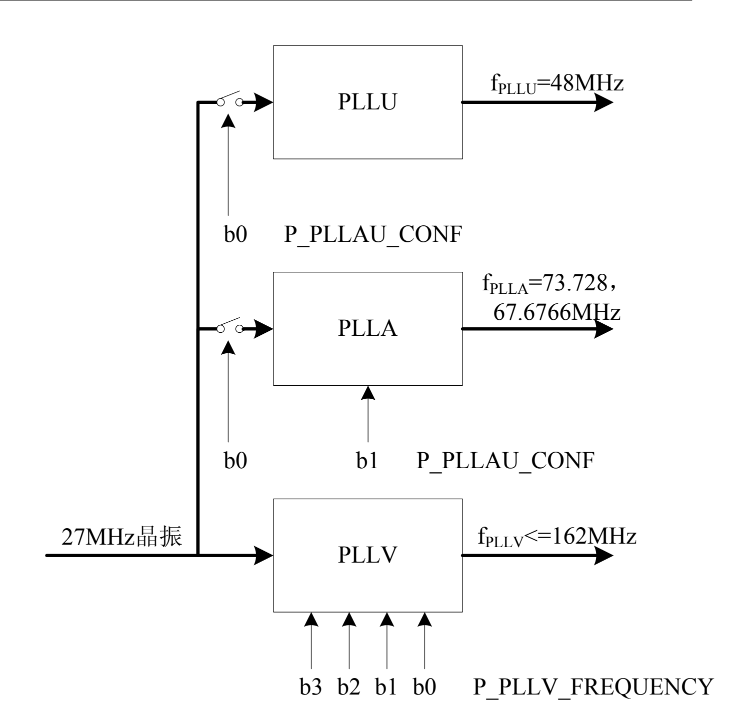


图 3.4 PLL 电路框图

经过 PLLV 倍频之后可以输出最大至 162MHz 的频率主要提供给 CPU 及 AHB 总线和外设使用。

PLLA 电路将 27MHz 倍频至 67.6766MHz 或 73.728MHz。

PLLU 电路为 USB 等模块提供 48MHz 的时钟频率。

在系统上电后，仅 PLLV 电路处于工作状态并为片内包括 CPU、MIU（Memory Interface Unit——存储器接口单元）、APBDMA 和软件配置模块等在内的主要模块提供工作时钟。当用户需要使用某个外设时，必须为该外设接通工作时钟。SPCE3200 内部的 PLL 控制器控制着所有外设的工作时钟的通断，表 3.3 描述了每个片内外设的工作时钟通断控制寄存器。

表 3.3 SPCE3200 外设工作时钟通断控制寄存器

寄存器地址	外设名称	寄存器地址	外设名称
0x88210018	CSI	0x88210070	TIMER1
0x88210020	TV 编码器	0x88210074	TIMER2
0x88210034	LCD 驱动器	0x88210078	TIMER3
0x882100D4	EPP	0x8821007C	TIMER4
0x88210058	APBDMA	0x88210080	TIMER5
0x882100C8	缓冲区控制器	0x88210088	RTC
0x882100E0	TimeBase	0x88210084	看门狗
0x882100CC	LDMDMA	0x882100A4	SD 卡控制器

寄存器地址	外设名称	寄存器地址	外设名称
0x88210050	BLNDMA	0x882100A8	Nand 型 Flash 控制器
0x88210098	SPI 控制器	0x882100AC	A/D 转换器
0x882100A0	SIO 控制器	0x88210064	USB
0x88210094	I2C 控制器	0x882100FC	软件配置控制器
0x8821008C	I2S 控制器	0x882100EC	MPEG4 编解码控制器
0x8821005C	UART	0x882100BC	PLLA&PLLU
0x8821006C	TIMER0	0x88210114	32768 晶振

3.5.2 时钟发生器 CKG

经过三个 PLL 电路倍频之后的时钟信号被送至时钟发生器单元，并在这里将时钟分频，分配给 SPCE3200 各个片上模块使用。其中，AHB 总线及其外设由 PLLV 或 PLLA 经过 CKG 电路分频提供工作频率，均工作在相同的频率下；APB 总线及其外设，除 A/D 转换器和看门狗模块外，均工作于 27MHz 频率下，定时器模块也可以使用 32768Hz 实时时钟晶振提供工作频率；A/D 转换器可以选择由 PLLU 或 PLLA 经过 CKG 电路分频提供工作频率；看门狗则由 32768Hz 时钟晶振提供工作频率。

CKG 时钟发生器单元为可以选择工作频率的某些模块提供了相应的控制寄存器，表 3.4 描述了这些模块的工作频率选择寄存器。

表 3.4 SPCE3200 外设工作时钟选择寄存器

寄存器地址	外设名称	寄存器地址	外设名称
0x88210004	CPU	0x882100B0	ADC
0x8821000C	MIU	0x882100B8	PLLV
0x8821001C	CSI	0x882100BC	PLLAU
0x8821002C	JPEG	0x882100DC	Sleep
0x88210038	LCD	0x882100E4	Timer
0x88210090	I2S	0x882100F0	MPEG4

3.5.3 寄存器描述

与各个模块相关的时钟通断控制寄存器和频率选择寄存器请参考各个功能部件章节，这里仅介绍 CPU、AHB 总线、PLLV、PLLA、PLLU 以及 32768Hz 实时时钟晶振等相关的时钟配置寄存器。

CPU 时钟选择寄存器：P_CLK_CPU_SEL(0x88210004)

用户可以通过此寄存器选择 CPU 的工作频率。注意，在系统上电后该寄存器不可直接修改，如果需要更改 CPU 时钟，请参考 3.5.4 节的操作方法。

表 3.5 P_CLK_CPU_SEL(0x88210004)

位	b3-b0
读/写	R/W
默认值	0
名称	CPU_CLOCK

CPU_CLOCK　　b3-b0　　CPU 时钟选择位：

0000：PLLV 时钟频率输出
0001：PLLV 时钟频率输出 2 分频
0010：PLLV 时钟频率输出 3 分频
0011：PLLV 时钟频率输出 4 分频
0100：PLLV 时钟频率输出 6 分频
0101：PLLV 时钟频率输出 8 分频
0110：PLLA 时钟频率输出
0111：PLLA 时钟频率输出 2 分频
1000：PLLA 时钟频率输出 3 分频
1001：PLLA 时钟频率输出 4 分频
1010：PLLA 时钟频率输出 6 分频
1011：PLLA 时钟频率输出 8 分频

AHB 总线时钟配置寄存器：P_CLK_AHB_CONF(0x88210008)

用户可以通过此寄存器打开或关闭 AHB 总线的工作时钟。注意，在系统上电后该寄存器不可直接修改，如果需要停止 AHB 总线时钟，请参考 3.5.4 节的操作方法。

表 3.6 P_ CLK_AHB_CONF(0x88210008)

位	b0
读/写	W
默认值	1
名称	CPUAHB_CLK_Stop

CPUAHB_CLK_Stop　b0　　暂时关闭 AHB 总线工作时钟，以便更改其工作频率

0：关闭 AHB 时钟

1：不关闭 AHB 时钟

AHB 总线时钟选择寄存器：P_CLK_AHB_SEL(0x8821000C)

用户可以通过此寄存器选择 AHB 总线的工作频率。注意，在系统上电后该寄存器不可直接修改，如果需要更改 AHB 总线时钟，请参考 3.5.4 节的操作方法。

表 3.7 P_CLK_AHB_SEL(0x8821000C)

位	b1~b0
读/写	W
默认值	0
名称	CPUAHB_CLK

CPUAHB_CLK　b1~b0　　AHB 时钟选择位：

00：CPU 工作频率

01：CPU 工作频率 2 分频

10：CPU 工作频率 3 分频

11：CPU 工作频率 4 分频

PLLV 时钟配置寄存器：P_CLK_PLLV_CONF(0x882100B4)

用户可以通过此寄存器打开或关闭 PLLV 的输出。注意，在系统上电后该寄存器不可直接修改，如果需要停止 PLLV 时钟，请参考 3.5.4 节的操作方法。

表 3.8 P_CLK_PLLV_CONF(0x882100B4)

位	b0
读/写	W
默认值	1
名称	PLLV_EN

PLLV_EN　b0　　PLLV 频率输出使能位：

0：关闭 PLLV 频率输出

1：打开 PLLV 频率输出

PLLV 时钟选择寄存器：P_CLK_PLLV_SEL(0x882100B8)

用户可以通过此寄存器选择 PLLV 的输出频率。注意，在系统上电后该寄存器不可直接修改，如果需要更改 PLLV 的时钟输出，请参考 3.5.4 节的操作方法。

表 3.9 P_CLK_PLLV_SEL(0x882100B8)

位	b3-b0
读/写	W
默认值	0
名称	FREQUENCY

FREQUENCY　　b3~b0　　PLLV 输出频率选择位：

0000：保留

0001：保留

0010：保留

0011：81MHz

0100：87.75MHz

0101：97.5MHz

0110：101.25MHz

0111：108MHz

1000：114.75MHz

1001：121.5MHz

1010：128.25MHz

1011：135MHz

1100：141.75MHz

1101：148.5MHz

1110：155.25MHz

1111：162MHz

PLLAU 时钟配置寄存器：P_CLK_PLLAU_CONF(0x882100BC)

用户可以通过此寄存器打开或关闭 PLLA 和 PLLU 的输出，并可以选择 PLLA 的输出频率。

表 3.10 P_CLK_PLLAU_CONF(0x882100BC)

位	b2	b1	b0
读/写	W	W	W
默认值	0	0	0
名称	USBPLL_EN	PLLA_CLK	PLLA_EN

USBPLL_EN　　b2　　PLLU 频率输出使能位：

0：关闭 PLLU 频率输出

1：打开 PLLU 频率输出

PLLA_CLK　　b1　　PLLA 输出频率选择位：

0：73.728MHz

1：67.6766MHz

PLLA_EN　　b0　　PLLA 频率输出使能位：

0：关闭 PLLA 频率输出

1：打开 PLLA 频率输出

32K 实时时钟配置寄存器：P_CLK_32K_CONF(0x88210114)

用户可以通过此寄存器使能或禁止 32768Hz 实时时钟晶振。

表 3.11 P_CLK_32K_CONF(0x88210114)

位	b0
读/写	W
默认值	0
名称	Crystal_En

Crystal_En　　b0　　32768Hz 实时时钟晶振使能位：

0：禁止

1：使能

3.5.4 系统时钟调整

停止 CPU/MIU 时钟以及 PLLV 时钟

在停止 MIU 时钟之前，必须首先使 DRAM 进入自刷新模式，否则，在停止 MIU 时钟之后存储在 DRAM 中的数据将丢失。

如果希望 CPU 进入睡眠模式，必须检查中断请求状态寄存器 P_INT_REQ_STATUS1 和 P_INT_REQ_STATUS2，清除所有可以唤醒 CPU 的中断，否则，CPU 进入睡眠模式后将立即被终止。

SPCE3200 内部 ROM 提供了一些程序入口来使 CPU、MIU 或 PLLV 时钟停止以便降低功耗。这些程序的入口地址如表 3.12 所示。

表 3.12 停止 CPU/MIU/PLLV 时钟 API 函数列表

程序入口	功能描述
0x9F000010	关闭 CPU 时钟
0x9F000014	关闭 CPU 和 MIU 时钟
0x9F000018	关闭 CPU、MIU 和 PLLV 时钟

用户需要使用 S+core 汇编编写下面的代码以便可以调用这些函数更改系统时钟。

```
@jumpto_extrom_discpu_clk
.global jumpto_extrom_discpu_clk
jumpto_extrom_discpu_clk:
    li r8, 0x9f000010
    br r8

@jumpto_extrom_discpu_miu_clk
.global jumpto_extrom_discpu_miu_clk
jumpto_extrom_discpu_miu_clk:
    li r8, 0x9f000014
    br r8

@jumpto_extrom_discpu_miu_clk_pll
.global jumpto_extrom_discpu_clk_pll
jumpto_extrom_discpu_clk_pll:
    li r8, 0x9f000018
```

```
    br r8
```

更改 CPU/MIU 时钟以及 PLLV 时钟

当希望更改 MIU 的时钟时，程序不能在 DRAM 中运行，因为 SPCE3200 将通过 MIU 模块来访问 DRAM 中的内容，一旦开始更改 MIU 的时钟，那么 MIU 中 DRAM 的参数设置将发生错误，此时 MIU 将不能正确访问 DRAM 的内容，从而导致程序运行出错。

用户必须使用 SPCE3200 内部 ROM 中提供的程序来更改 CPU/MIU 的时钟，程序入口地址为 0x9F000024。

注意，更改 CPU 时钟的程序将对 PLLV 和 PLLA 进行操作，所以，在执行更改 CPU 时钟的程序之前，必须检查芯片内使用 PLLV 和 PLLA 的模块没有处于工作状态。在更改完毕 CPU 时钟后，PLLA 将处于打开状态，如果不需要使用它，用户可以将其关闭。

另外需要注意，CPU 的时钟必须是 MIU 时钟的 1～4 倍。

SPCE3200 内部 ROM 提供的更改 CPU 时钟的程序需要用户传递四个参数以便指定更改后的时钟频率。第一个参数含有七个子项，含义如表 3.13 所示。

表 3.13 更改时钟频率程序的第一个参数

子项名称	有效位	影响寄存器	功能描述
	b31~b12	无	保留
PLLA_Enable	b11	P_CLK_PLLAU_CONF	PLLA 使能位： 0：使能 1：不使能
PLLV_Freq_Chg	b10	P_CLK_PLLV_SEL	PLLV 是否改变： 0：不改变 1：改变
AHBSLV_Sel	b9~b8	P_CLK_AHB_SEL	MIU 时钟选择位： 00：CPU 工作频率 01：CPU 工作频率 2 分频 10：CPU 工作频率 3 分频 11：CPU 工作频率 4 分频
PLL_DIV_Setting	b7~b4	P_CLK_CPU_SEL	CPU 时钟选择位： 0000：PLLV 时钟频率输出 0001：PLLV 时钟频率输出 2 分频 0010：PLLV 时钟频率输出 3 分频

			0011：PLLV 时钟频率输出 4 分频 0100：PLLV 时钟频率输出 6 分频 0101：PLLV 时钟频率输出 8 分频 0110：PLLA 时钟频率输出 0111：PLLA 时钟频率输出 2 分频 1000：PLLA 时钟频率输出 3 分频 1001：PLLA 时钟频率输出 4 分频 1010：PLLA 时钟频率输出 6 分频 1011：PLLA 时钟频率输出 8 分频
PLLV_Freq_Setting	b3~b0	P_CLK_PLLV_SEL	PLLV 输出频率选择位： 0000：保留 0001：保留 0010：保留 0011：81MHz 0100：87.75MHz 0101：97.5MHz 0110：101.25MHz 0111：108MHz 1000：114.75MHz 1001：121.5MHz 1010：128.25MHz 1011：135MHz 1100：141.75MHz 1101：148.5MHz 1110：155.25MHz 1111：162MHz

第二个参数用来设置 SDRAM 的参数，含义如表 3.14 所示。

表 3.14 更改时钟频率程序的第二个参数

子项名称	有效位	影响寄存器	功能描述
	b31~b17	无	保留

MIU_Run108_Sel	b16	P_MIU_SDRAM_SETUP1	MIU 时钟是否运行在 108MHz： 0：不运行 1：运行
MIU_Timing_Setting	b15~b0	P_MIU_SDRAM_SETUP1	MIU 时钟频率选择位： 0x48C0：27MHz 0x4920：40MHz 0x4983：54MHz 0x4A03：67.5MHz 0x4A03：81MHz 0x4B04：108MHz

第三个和第四个参数用来指定 SPCE3200 改变时钟频率后等待 PLLV 变为稳定的时钟周期。

用户可以编写下面所示的 C 代码和汇编代码以便调用该程序来改变时钟频率。

C 代码：

```
    extern void jumpto_extrom_change_cpuclk(unsigned int r4, unsigned int r5, unsigned int r6, unsigned
int r7);
    void PLL_change_func(
        int PLLV_Freq_Setting,
        int PLL_DIV_Setting,
        int AHBSLV_SEL,
        int PLLV_Freq_Chg,
        int PLLA_Enable,
        int MIU_Timing_Setting,
        int MIU_Run108_Sel
    ){
        unsigned int r4, r5, r6, r7;
        r4 = PLLV_Freq_Setting + (PLLV_DIV_Setting << 4) + (AHBSLV_SEL << 8) +
            (PLLV_Freq_Chg << 10) + (PLLA_Enable << 11);
        r5 = (MIU_Run108_Sel << 16) + MIU_Timging_Setting;

        r6 = 0xff;
        r7 = 0xff;
```

```
        jumpto_extrom_change_cpuclk(r4, r5, r6, r7);
}
```

汇编代码：

```
    @jumpto_extrom_change_cpuclk
    .global jumpto_extrom_change_cpuclk
    jumpto_extrom_change_cpuclk:
        ldis r8, 0x9f00
        ori r8, 0x0024
        br r8
```

3.6 中断控制器

3.6.1 概述

为了最大效率的使用 CPU 资源，SPCE3200 也引入了中断机制。SPCE3200 通过中断控制器向 CPU 发出中断请求，同时，当有多个中断发生时，中断控制器进行中断优先级的仲裁。

SPCE3200 的中断控制器为 40 个中断源提供了屏蔽控制能力，并将这些中断源进行处理、向处理器发出中断请求信号。中断控制器的作用是对来自内部外设和外部中断请求管脚上发生的多个中断请求进行仲裁处理(优先权排队)后，然后依处理结果向 CPU 发出 IRQ 中断请求。

通常，CPU 只接收来自外设的 IRQ 中断事件，但这种中断接收并没有优先级机制。因此，中断处理程序必须借助中断控制器里的硬件来处理这种情况。譬如，假设所有中断源都已设置成 IRQ 中断，其中有 10 个中断源向 CPU 同时发出中断请求，那么，就可以通过软件读出中断控制器里的中断状态寄存器来获知都发生了哪些中断请求，从而确定中断服务的优先级。

可想而知，这种中断处理需要较长的中断响应时间才能跳转到相应的服务程序的位置上，为此，S+core 处理器提供了另一种中断处理机制，称为向量中断模式，它是 RISC 型微处理器所共有的特点。当有多个中断源发出中断请求时，由硬件优先权逻辑来确定应当先服务于哪一个中断请求；同时该硬件优先权逻辑将向量表中向量的基地址与偏移值相配，计算出相应服务程序位置的地址。S+core 处理器也相应地提供了一些指令，配合硬件来获取向量表的入口，并实现相应中断服务程序位置的跳转，从而大大缩短了中断响应时间。

由此，中断控制器的功能作用总结起来有三个：

（1）40 个中断源的屏蔽控制。

（2）中断优先级进的裁决。

（3）向 CPU 发出中断请求。

3.6.2 特性

SPCE3200 中断控制器的特性如下：

- 为易于实现 Soc 集成，总线符合 AMBA 规格(Rev 2.0)
- 支持 40 个中断源
- IRQ 中断为高电平有效
- 支持向量中断模式
- 每一中断源都可编程设置其优先级
- 每一个中断源的中断都可独立被屏蔽或者使能

3.6.3 中断源

SPCE3200 有 40 个中断源，占用 S+core 内核中第 24~63 个硬件中断向量，如表 3.15。

表 3.15 SPCE3200 的中断源列表

序号	组名	中断向量号	中断服务函数	中断源	助记名	中断向量地址=中断基址+
1	SLV0	63	IRQ63()	DAC 中断	DAC_IRQ	63×0x04(0x10)
2		62	IRQ62()	保留	RESERVED	62×0x04(0x10)
3		61	IRQ61()	保留	RESERVED	61×0x04(0x10)
4		60	IRQ60()	保留	RESERVED	60×0x04(0x10)
5		59	IRQ59()	MIC 溢出中断	MIC_OV	59×0x04(0x10)
6		58	IRQ58()	ADC 中断	ADC_IRQ	58×0x04(0x10)
7		57	IRQ57()	时基中断	TMB_IRQ	57×0x04(0x10)
8		56	IRQ56()	定时器中断	TIMER_IRQ	56×0x04(0x10)
9	SLV1	55	IRQ55()	TV 垂直空白开始中断	TV_VBS	55×0x04(0x10)
10		54	IRQ54()	LCD 垂直空白开始中断	LCD_VBS	54×0x04(0x10)
11		53	IRQ53()	保留	RESERVED	53×0x04(0x10)
12		52	IRQ52()	TV 光枪中断	TV_LIGHT	52×0x04(0x10)
13		51	IRQ51()	CSI 帧结束中断	CSI_FRE	51×0x04(0x10)
14		50	IRQ50()	CSI 坐标击中中断	CSI_COH	50×0x04(0x10)
15		49	IRQ49()	CSI 运动帧结束中断	CSI_MFRE	49×0x04(0x10)

序号	组名	中断向量号	中断服务函数	中断源	助记名	中断向量地址=中断基址+
16		48	IRQ48()	CSI 捕获完成中断	CSI_DONE	48×0x04(0x10)
17	SLV2	47	IRQ47()	TV 坐标击中中断	TV_COH	47×0x04(0x10)
18		46	IRQ46()	保留	RESERVED	46×0x04(0x10)
19		45	IRQ45()	USB 主从中断	USB_IRQ	45×0x04(0x10)
20		44	IRQ44()	SIO 中断	SIO_IRQ	44×0x04(0x10)
21		43	IRQ43()	SPI 中断	SPI_IRQ	43×0x04(0x10)
22		42	IRQ42()	UART 中断	UART_IRQ	42×0x04(0x10)
23		41	IRQ41()	NAND 中断	NAND_IRQ	41×0x04(0x10)
24		40	IRQ40()	SD 卡通讯中断	SD_IRQ	40×0x04(0x10)
25	SLV3	39	IRQ39()	I2C 主模式中断	I2C_IRQ	39×0x04(0x10)
26		38	IRQ38()	I2S 从模式中断	I2S_IRQ	38×0x04(0x10)
27		37	IRQ37()	APB DMA 通道 0 中断	APBD_CH0	37×0x04(0x10)
28		36	IRQ36()	APB DMA 通道 1 中断	APBD_CH1	36×0x04(0x10)
29		35	IRQ35()	LDM DMA 中断	LDM_DMA	35×0x04(0x10)
30		34	IRQ34()	BLN DMA 中断	BLN_DMA	34×0x04(0x10)
31		33	IRQ33()	APB DMA 通道 2 中断	APBD_CH2	33×0x04(0x10)
32		32	IRQ32()	APB DMA 通道 3 中断	APBD_CH3	32×0x04(0x10)
33	SLV4	31	IRQ31()	实时时钟中断	RTC_IRQ	31×0x04(0x10)
34		30	IRQ30()	MPEG4 中断	MPEG4_IRQ	30×0x04(0x10)
35		29	IRQ29()	ECC 中断	ECC_IRQ	29×0x04(0x10)
36		28	IRQ28()	GPIO 外部中断	GPIO_IRQ	28×0x04(0x10)
37		27	IRQ27()	TV 垂直空白结束中断	TV_VBE	27×0x04(0x10)
38		26	IRQ26()	保留	RESERVED	26×0x04(0x10)
39		25	IRQ25()	保留	RESERVED	25×0x04(0x10)
40		24	IRQ24()	保留	RESERVED	24×0x04(0x10)

说明：

（1）为了优先级设置方便，人为把 40 个中断源分成了 5 组，每组 8 个中断源，详见中断优先级设定寄存器。

（2）中断向量号是 CPU 处理各中断源中断服务程序的标识，根据中断向量号可以计算出中断服务程序的入口地址，详见中断机制章节。

（3）助记名是为了方便各寄存器介绍时约定的符号，详见寄存器章节。

（4）计算中断向量地址两种办法，即偏移量可以是 0x04，也可以是 0x10，用户可以通过设置 S+core 内核寄存器 cr3 的 bit0 设置其偏移量，见中断机制章节。

3.6.4 结构框图

SPCE3200 中断控制器的结构如图 3.5：在使用中断处理一功能模块的任务时，需要先通过模块的中断控制器使能该模块中断；中断控制器需要先使能（允许）来自该模块中断源的中断，当有中断请求时，中断控制器把中断状态寄存器的相应位置位，并把该请求送给中断优先级裁决器，中断优先级裁决器根据设定好的优先级决定向 CPU 发送相应模块的中断请求；时钟控制器控制中断控制器模块的复位。

图 3.5 中断控制器结构框图

3.6.5 寄存器描述

根据图 3.5 中断控制器结构框图可以看出，SPCE3200 中断控制器的硬件寄存器有时钟控制寄存器、中断源中断使能（允许）寄存器、中断请求状态寄存器和中断优先级设定寄存器四大类共 10 个寄存器，如表 3.16。

表 3.16 中断控制器相关寄存器列表

寄存器名称	助记符	地址
中断时钟配置寄存器	P_INT_CLK_CONF	0x882100D0
中断控制寄存器 1	P_INT_MASK_CTRL1	0x880A0020
中断控制寄存器 2	P_INT_MASK_CTRL2	0x880A0024
中断请求状态寄存器 1	P_INT_REQ_STATUS1	0x880A0000
中断请求状态寄存器 2	P_INT_REQ_STATUS2	0x880A0004
中断优先级寄存器	P_INT_GROUP_PRI	0x880A0008
第 0 组中断优先级寄存器	P_INT_GROUP0_PRI	0x880A0010
第 1 组中断优先级寄存器	P_INT_GROUP1_PRI	0x880A0014
第 2 组中断优先级寄存器	P_INT_GROUP2_PRI	0x880a0018
第 3 组中断优先级寄存器	P_INT_GROUP3_PRI	0x880A001C

中断时钟配置寄存器：P_INT_CLK_CONF(0x882100D0)

通过中断时钟配置寄存器可以复位/不复位中断控制器模块，正常使用时不需要复位中断控制器。

表 3.17 P_INT_CLK_CONF(0x882100D0)

位	**b31~b1**	**b0**
读/写	-	W
默认值	-	1
名称	-	IRQ_RST

IRQ _RST   b0   中断控制器模块复位位：

0：中断控制器模块复位

1：中断控制器模块不复位

中断控制寄存器 1：P_INT_MASK_CTRL1 (0x880A0020)

SPCE3200 有 40 个硬件中断源（中断向量 63~24），由两个 32 位寄存器 P_INT_MASK_CTRL1 和 P_INT_MASK_CTRL2 来控制这些中断源中断的使能或者屏蔽。

表 3.18　P_INT_MASK_CTRL1 (0x880A0020)

位	b31	b30	b29	b28	b27	b26	b25	b24
读/写	W	W	W	W	W	W	W	W
默认值	1	1	1	1	1	1	1	1
名称	APBD_CH3	APBD_CH2	BLN_DMA	LDM_DMA	APBD_CH1	APBD_CH0	I2S_IRQ	I2C_IRQ
位	b23	b22	b21	b20	b19	b18	b17	b16
读/写	W	W	W	W	W	W	W	W
默认值	1	1	1	1	1	1	1	1
名称	SD_IRQ	NAND_IRQ	UART_IRQ	SPI_IRQ	SIO_IRQ	USB_IRQ	RESERVED	TV_COH
位	b15	b14	b13	b12	b11	b10	b9	b8
读/写	W	W	W	W	W	W	W	W
默认值	1	1	1	1	1	1	1	1
名称	CSI_DONE	CSI_MFRE	CSI_COH	CSI_FRE	TV_LIGHT	RESERVED	LCD_VBS	TV_VBS
位	b7	b6	b5	b4	b3	b2	b1	B0
读/写	W	W	W	W	W	W	W	W
默认值	1	1	1	1	1	1	1	1
名称	TIMER_IRQ	TMB_IRQ	ADC_IRQ	MIC_OV	RESERVED	RESERVED	RESERVED	DAC_IRQ

APBD_CH3　b31　　APB DMA 通道 3 中断屏蔽位：

0：使能 APB DMA 通道 3 中断

1：屏蔽 APB DMA 通道 3 中断

APBD_CH2　b30　　APB DMA 通道 2 中断屏蔽位：

0：使能 APB DMA 通道 2 中断

1：屏蔽 APB DMA 通道 2 中断

……	……	……
ADC_IRQ	b5	ADC 中断屏蔽位： 0：使能 ADC 中断 1：屏蔽 ADC 中断
MIC_OV	b4	MIC 溢出中断屏蔽位： 0：使能 MIC 溢出中断 1：屏蔽 MIC 溢出中断
DAC_IRQ	b0	DAC 中断屏蔽位： 0：使能 DAC 中断 1：屏蔽 DAC 中断

说明：

（1）表中寄存器所有 32 位的功能一样，写“1”屏蔽该位对应的中断，写“0”使能该位对应的中断。

（2）各位对应的中断名称请参考表 3.15。

中断控制寄存器 2：P_INT_MASK_CTRL2(0x880A0024)

通过中断控制寄存器 2 使能或者屏蔽来自中断向量 31~24 中断源请求的中断。

表 3.19 P_INT_MASK_CTRL2(0x880A0024)

位	b7	b6	b5	b4	b3	b2	b1	b0
读/写	W	W	W	W	W	W	W	W
默认值	1	1	1	1	1	1	1	1
名称	RESERVED	RESERVED	RESERVED	TV_VBE	GPIO_IRQ	ECC_IRQ	MP4_IRQ	RTC_IRQ

与 P_INT_MASK_CTRL1 类似，P_INT_MASK_CTRL2 的低 8 位中，各位的功能相同：写“1”屏蔽该位对应的中断，写“0”使能该位对应的中断。各位对应的中断名称请参考表 3.15。

SPCE3200 由两个 32 位硬件寄存器 P_INT_REQ_STATUS1 和 P_INT_REQ_STATUS2 来反映 40 个中断源的请求状态。

中断请求状态寄存器 1：P_INT_REQ_STATUS1(0x880A0000)

通过中断请求状态寄存器 1 来获得中断向量 63~32 中断源的中断请求状态。如果中断控制器收到中断源的中断请求，则该中断源对应的位被置位。

表 3.20　P_INT_REQ_STATUS1(0x880A0000)

位	b31	b30	b29	b28	b27	b26	b25	b24
读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	APBD_CH3	APBD_CH2	BLN_DMA	LDM_DMA	APBD_CH1	APBD_CH0	I2S_IRQ	I2C_IRQ
位	b23	b22	b21	b20	b19	b18	b17	b16
读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	SD_IRQ	NAND_IRQ	UART_IRQ	SPI_IRQ	SIO_IRQ	USB_IRQ	RESERVED	TV_COH
位	b15	b14	b13	b12	b11	b10	b9	b8
读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	CSI_DONE	CSI_MFRE	CSI_COH	CSI_FRE	TV_LIGHT	RESERVED	LCD_VBS	TV_VBS
位	b7	b6	b5	b4	b3	b2	b1	B0
读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	TIMER_IRQ	TMB_IRQ	ADC_IRQ	MIC_OV	RESERVED	RESERVED	RESERVED	DAC_IRQ

APBD_CH3　b31　APB DMA 通道 3 中断请求状态位：

0：APB DMA 通道 3 没有向 CPU 发出中断请求

1：APB DMA 通道 3 向 CPU 发出中断请求

APBD_CH2　b30　APB DMA 通道 2 中断请求状态位：

0：APB DMA 通道 2 没有向 CPU 发出中断请求

1：APB DMA 通道 2 向 CPU 发出中断请求

……　……　……

ADC_IRQ　b5　ADC 中断请求状态位：

0：ADC 没有向 CPU 发出中断请求

1：ADC 向 CPU 发出中断请求

MIC_OV　b4　MIC 溢出中断请求状态位：

0：MIC 溢出没有向 CPU 发出中断请求

1：MIC 溢出向 CPU 发出中断请求

DAC_IRQ　　b0　　　DAC 中断请求状态位：

0：DAC 没有向 CPU 发出中断请求

1：DAC 向 CPU 发出中断请求

说明：

（1）表中寄存器所有 32 位的功能一样，读为“1”表示产生了中断请求，读为“0”表示没有产生中断请求。

（2）各位对应的中断名称请参考表 3.15。

中断请求状态寄存器 2：P_INT_REQ_STATUS2(0x880A0004)

通过中断请求状态寄存器 2 来获得中断向量 31~24 中断源的中断请求状态。

表 3.21　P_INT_REQ_STATUS2(0x880A0004)

位	b7	b6	b5	b4	b3	b2	b1	b0
读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	RESERVED	RESERVED	RESERVED	TV_VBE	GPIO_IRQ	ECC_IRQ	MP4_IRQ	RTC_IRQ

与 P_INT_REQ_STATUS1 类似，P_INT_REQ_STATUS2 的低 8 位中，各位的功能相同：读为“1”表示产生了中断请求，读为“0”表示没有产生中断请求。各位对应的中断名称请参考表 3.15。

中断优先级寄存器：P_INT_GROUP_PRI(0x880A0008)

SPCE3200 的中断向量 63~32 的中断优先级可通过寄存器设置，中断向量 31~24 的中断优先级由中断控制器硬件控制。由于 32 个中断源的中断优先级不能通过一个寄存器来设定，所以先把这 32 个中断源分为 4 组，分组情况参考表 3.15，每组有 8 个中断源，先分别设置各组的优先级，然后设置各组内部各中断的优先级。

表 3.22　P_INT_GROUP_PRI(0x880A0008)

位	b7	b6	b5	b4	b3	b2	b1	b0
读/写	W	W	W	W	W	W	W	W
默认值	1	1	1	0	0	1	0	0

名称	P_SLV3	P_SLV2	P_SLV1	P_SLV0

P_SLV3 b7~b6 第 3 组中断的优先级设置位：

00：第 1 优先级

01：第 2 优先级

10：第 3 优先级

11：第 4 优先级

P_SLV2 b5~b4 第 2 组中断的优先级设置位：

00：第 1 优先级

01：第 2 优先级

10：第 3 优先级

11：第 4 优先级

P_SLV1 b3~b2 第 1 组中断的优先级设置位：

00：第 1 优先级

01：第 2 优先级

10：第 3 优先级

11：第 4 优先级

P_SLV0 b1~b0 第 0 组中断的优先级设置位：

00：第 1 优先级

01：第 2 优先级

10：第 3 优先级

11：第 4 优先级

第 0 组中断优先级寄存器：P_INT_GROUP0_PRI(0x880A0010)

通过 P_I_PSLV0 寄存器控制第 0 组中断中各中断的优先级。

表 3.23 P_INT_GROUP0_PRI(0x880A0010)

位	b23~b21	b20~b18	b17~b15	b14~b12	b11~b9	b8~b6	b5~b3	b2~b0
读/写	W	W	W	W	W	W	W	W
默认值	111b	110b	101b	100b	011b	010b	001b	000b
名称	TIMER_IRQ	TMB_IRQ	ADC_IRQ	MIC_OV	RESERVED	RESERVED	RESERVED	DAC_IRQ

从 b0~b23 共 24 位控制 8 个中断的优先级，每三位控制一个中断的优先级，三位从高到低的组合（b23b22b21、b20b19b18、b17b16b15、b14b13b12、b11b10b9、b8b7b6、b5b4b3、b2b1b0）功能如下：

000：第 1 优先级。

001：第 2 优先级。

010：第 3 优先级。

011：第 4 优先级。

100：第 5 优先级。

101：第 6 优先级。

110：第 7 优先级。

111：第 8 优先级。

默认最低 3 位（b2~b0）对应的中断优先级最高，最高 3 位（b23~b21）对应的中断优先级最低。

表中对应的中断名称可参考表 3.15。

寄存器 P_INT_GROUP1_PRI、P_INT_GROUP2_PRI、P_INT_GROUP3_PRI 和 P_INT_GROUP0_PRI 类似，只是寄存器中每三位对应的中断源各不相同而已。

第 1 组中断优先级寄存器：P_INT_GROUP1_PRI(0x880A0014)

通过 P_INT_GROUP1_PRI 寄存器控制第 1 组各中断源的优先级。

表 3.24 P_INT_GROUP1_PRI(0x880A0014)

位	b23~b21	b20~b18	b17~b15	b14~b12	b11~b9	b8~b6	b5~b3	b2~b0
读/写	W	W	W	W	W	W	W	W
默认值	111b	110b	101b	100b	011b	010b	001b	000b
名称	CSI_DONE	CSI_MFRE	CSI_COH	CSI_FRE	TV_LIGHT	RESERVED	LCD_VBS	TV_VBS

第 2 组中断优先级寄存器：P_INT_GROUP2_PRI(0x880A0018)

通过 P_INT_GROUP2_PRI 寄存器控制第 2 组各中断源的优先级。

表 3.25 P_INT_GROUP1_PRI(0x880A0014)

位	b23~b21	b20~b18	b17~b15	b14~b12	b11~b9	b8~b6	b5~b3	b2~b0
读/写	W	W	W	W	W	W	W	W

默认值	111b	110b	101b	100b	011b	010b	001b	000b
名称	SD_IRQ	NAND_IRQ	UART_IRQ	SPI_IRQ	SIO_IRQ	USB_IRQ	RESERVED	TV_COH

第 3 组中断优先级寄存器：P_INT_GROUP3_PRI(0x880A001C)

通过 P_INT_GROUP3_PRI 寄存器控制第 3 组各中断源的优先级。

表 3.26 P_INT_GROUP1_PRI(0x880A0014)

位	b23~b21	b20~b18	b17~b15	b14~b12	b11~b9	b8~b6	b5~b3	b2~b0
读/写	W	W	W	W	W	W	W	W
默认值	111b	110b	101b	100b	011b	010b	001b	000b
名称	APBD_CH3	APBD_CH2	BLN_DMA	LDM_DMA	APBD_CH1	APBD_CH0	I2S_IRQ	I2C_IRQ

3.6.6 中断机制

SPCE3200 的中断机制是采用中断向量的模式，每个中断向量对应一个中断源（中断源列表参考表 4.52），但是每个中断源不一定对应一个中断，比如 UART 中断源分别有发送中断和接收中断。中断的处理过程如图 3.6。中断的处理过程涉及的硬件单元包括：外设单元（ADC、Timer 等）、中断控制器和 S+core 内核。

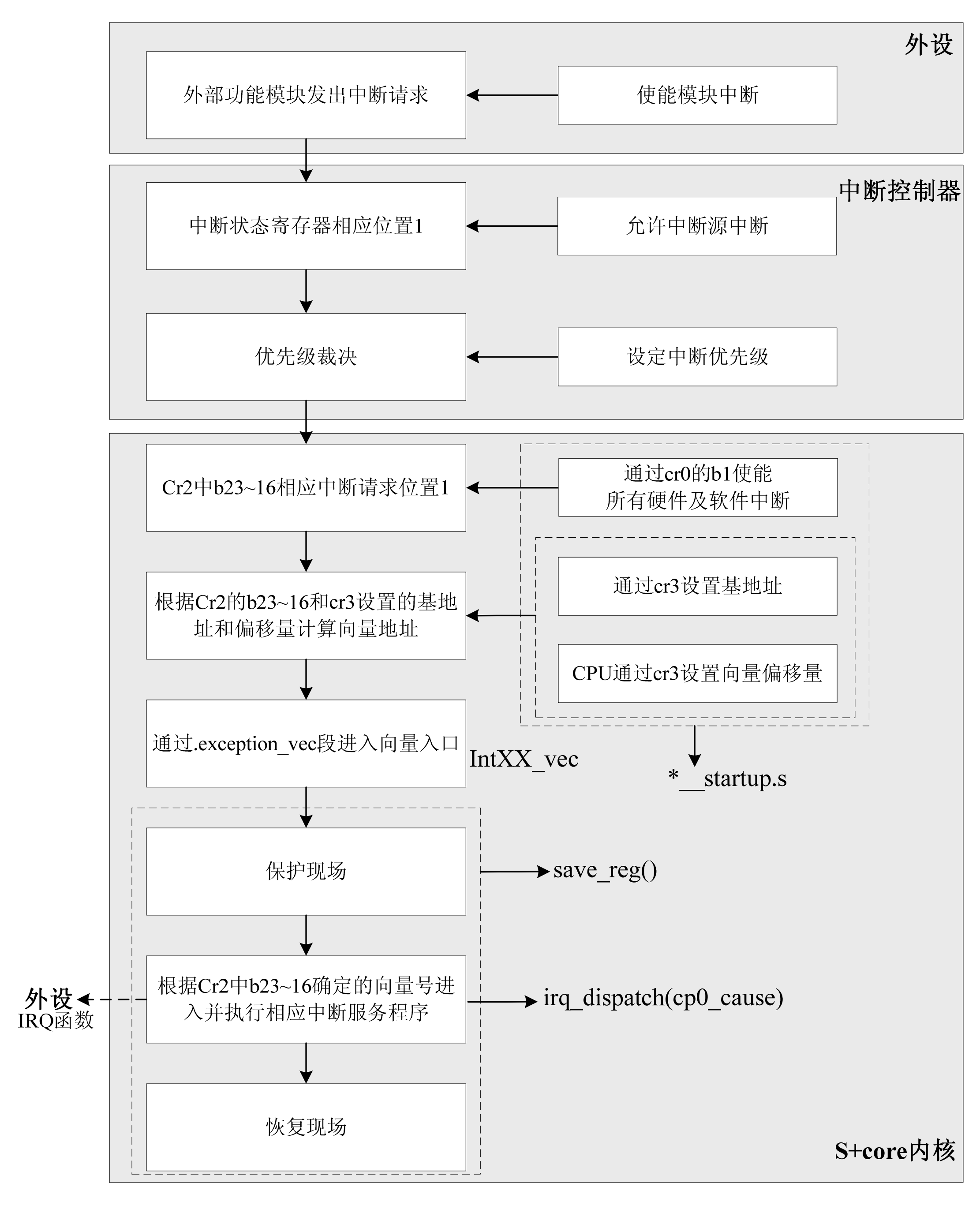


图 3.6　中断的处理过程图

图 3.6 中外设单元部分主要负责使能硬件模块中断和发出中断请求，必须通过相应寄存器使能模块中断，该模块才会发出中断请求。比如先要通过 0x88160000 地址使能 Timer0 中断，Timer 才会在设定定时时间时发出中断请求。

中断控制器主要负责中断源的屏蔽和中断优先级的裁决，在外设发出中断请求前，中断控制器必须要先关闭该模块对应中断源的中断屏蔽，这样中断控制器才会做优先级裁决并发出中断请求；如果在外设发出中断请求前，该中断源的中断是被屏蔽的，则在外设发出中断请求时中断控制器不做任何处理；中断控制器有专门的寄存器可设置中断优先级，中断控制器按照设置好的优先级进行优先级裁决。

S+core 内核必须允许来自模块中断源的中断，才会响应中断；通过程序状态寄存器 PSR（cr0）允许 63 个（含 SPCE3200 的 40 个中断源）硬件中断。S+core 在通过异常原因寄存器 ECR（cr2）得到中断请求相应位置 1 时，通过 cr2 的 b23~b18 得到向量号，并通过向量号、基址和偏移量得到其向量入口地址；通过 cr3 设置中断的基址，其中最低位设置偏移量，当最低位为 0 时，偏移量为 0x04，当最低位为 1 时，偏移量为 0x10；比如通过异常向量寄存器 EVCPVec（cr3）设置中断的基址为 0xa0000000，偏移量为 0x04，中断段的起始地址为 0xa00001fc，例如 Timer 请求中断的向量号为 56，那么其中断向量地址为 0xa0000200+4×56。在 S+core 找到中断向量入口地址后即可转到相应的中断服务程序执行，控制外设进行相应操作，如图 3.6 虚线箭头部分。

图 3.6 中关于中断的各个过程分别在*_startup.s、Sys_isr.s、Sys_IRQ.c 和 User_IRQ.c 四个文件中进行设置或者定义，这些文件在建立工程时可根据用户要求自动生成。

_startup.s（“”为工程名）定义了 S+core 内核的所有硬件中断的使能、中断基址的设置和偏移量的设置。中断相关程序段如下：

```
//==============================================================
// 中断使能
//==============================================================
    li r4, 0x1
    mtcr r4, cr0          // 使能所有 63 个硬件中断和 2 个软件中断
    li r4, 0xa0000000
    mtcr r4, cr3          // 指定异常基地址为 0xa0000000,并指定向量地址的偏移模式为偏移 0x4
```

注意：cr0 和 cr3 各位功能可参考第 1 章 S+core 内核寄存器。

在*_startup.s 文件中指定了异常的基地址，另外*_Prog.ld 定义了包括可以定义包括.DATA、.TEXT、异常段等所有工程中用到的段，*_Prog.ld 也是在建立工程时会自动生成的文件，异常段定义程序如下：

```
.exception 0xA00001fc :      // 定义中断异常段，起始地址为 0xA00001fc
{
    *(.exception_vec)        // 异常段名为“.exception_vec”，该段在 Sys_isr.S 文件中定义
} = 0
```

Sys_isr.s 定义个各中断向量的入口（IntX_vec）和调用中断服务函数（irq_dispatch(cp0_cause)）；其中 IntX_vec 中的 X=1,2,3…62，63。

参考以下程序段：由于.exception_vec 异常段的起始地址为 0xa00001fc，两个软件地址各占去 4 个字节地址单元（字对齐），所以 63 个硬件中断向量的入口地址为 0xa0000200+（0x04*n），其中 0x04 为偏移量，n 为 1，2，3…，62，63。

```
//==========================================================
// 文 件 名: Sys_isr.S
// 功能描述: 中断向量处理文件
//              使用时，用户不需要修改此文件中的内容
//==========================================================
#define     SP          r0
.section .exception_vec,"ax"
//定义异常中断段，在*_Prog.ld 中已经定义了其起始地址为 0xa00001fc
//==========================================================
// 定义两个软件中断向量入口
//==========================================================
.align 2                        // 强制 norm_debug_vec 的地址为字对齐，
                                // 即 norm_debug_vec 的地址为 0xa00001fc
norm_debug_vec:                 // Normal debug 异常向量入口
    j    norm_debug_service
.align 2                        // 强制 general_vec 的地址为字对齐，
                                // 即 general_vec 的地址为 0xa0000200
general_vec:                    // General 异常向量入口
    j    general_service
//==========================================================
// 定义 63 个硬件中断向量入口
//==========================================================
.align 2                        // int1_vec 的地址为 0xa0000200+（0x04*1）
int1_vec:                       // 中断向量 IRQ1 入口
    j    int_service            // 调用 int_service 函数
.align 2                        // int2_vec 的地址为 0xa0000200+（0x04*2）
int2_vec:                       // 中断向量 IRQ2 入口
    j    int_service            // 调用 int_service 函数
……
.align 2                        // int24_vec 的地址为 0xa0000200+（0x04*24）
int24_vec:                      // (Reserved)中断向量 IRQ24 入口
    j    save_reg               // 调用 save_reg 函数取得中断向量号
```

```
.align 2                        //   int25_vec 的地址为 0xa0000200+（0x04*25）
int25_vec:                      // (Reserved)中断向量 IRQ25 入口
    j     save_reg              //  调用 save_reg 函数取得中断向量号
……
.align 2                        // int56_vec 的地址为 0xa0000200+（0x04*56）
Int56_vec:                      // (Timer)中断向量 IRQ56 入口
    j     save_reg              //  调用 save_reg 函数取得中断向量号
……
.align 2                        // int62_vec 的地址为 0xa0000200+（0x04*62）
int62_vec:                      // (Reserved)中断向量 IRQ62 入口
    j     save_reg              //  调用 save_reg 函数取得中断向量号
.align 2                        // int63_vec 的地址为 0xa0000200+（0x04*63）
int63_vec:                      // (DAC)中断向量 IRQ63 入口
    j     save_reg              //  调用 save_reg 函数取得中断向量号

.extern intmsg                  //  该函数在 Sys_IRQ.c 文件中定义
//=============================================================
//  软件中断服务函数
//=============================================================
norm_debug_service:             //  软件中断 1(Normal debug 异常)中断服务程序
    jl intmsg                   //  调用 intmsg 函数

general_service:                //  软件中断 2(General 异常)中断服务程序
    jl intmsg                   //  调用 intmsg 函数

int_service:
    jl intmsg                   //  调用 intmsg 函数

.extern irq_dispatch            //  该函数在 Sys_IRQ.c 文件中定义
.set r1
//=============================================================
//  汇编语言格式:  save_reg
```

```
// C 语 言 格式:  void save_reg(void)
// 功 能  描 述:  中断服务函数，
//                保护及恢复现场，调用 irq_dispatch 函数
// 入 口  参 数:  无
// 出 口  参 数:  无
//==========================================================
save_reg:
    // 保存现场：保存 r1~r31 寄存器,保存条件寄存器 cr1 和中断计数器 cr5
    subi SP, 38*4
    sw   r1, [SP,1*4]
    sw   r2, [SP,2*4]
    sw   r3, [SP,3*4]
    sw   r4, [SP,4*4]
    sw   r5, [SP,5*4]
    sw   r6, [SP,6*4]
    sw   r7, [SP,7*4]
    sw   r8, [SP,8*4]
    sw   r9, [SP,9*4]
    sw   r10, [SP,10*4]
    sw   r11, [SP,11*4]
    sw    r12, [SP,12*4]
    sw   r13, [SP,13*4]
    sw   r14, [SP,14*4]
    sw   r15, [SP,15*4]
    sw   r16, [SP,16*4]
    sw   r17, [SP,17*4]
    sw   r18, [SP,18*4]
    sw   r19, [SP,19*4]
    sw   r20, [SP,20*4]
    sw   r21, [SP,21*4]
    sw   r22, [SP,22*4]
    sw   r23, [SP,23*4]
```

```
sw   r24, [SP,24*4]
sw   r25, [SP,25*4]
sw   r26, [SP,26*4]
sw   r27, [SP,27*4]
sw   r28, [SP,28*4]
sw   r29, [SP,29*4]
sw   r30, [SP,30*4]
sw   r31, [SP,31*4]
mfcr     r13, cr1
mfcr     r15, cr5
sw   r13, [SP,33*4]
sw   r15, [SP,35*4]

mfcr     r4, cr2
// 通过 cr2 的 b23~b16 读取中断请求的向量号，并把此数据作为调用函数的入口参数
jl    irq_dispatch    // 调用 irq_dispatch 函数，此函数在 Sys_IRQ.c 中定义

// 恢复现场：恢复 r1~r31 寄存器,条件寄存器 cr1 和中断计数器 cr5 中断前的数据
lw   r1, [SP,1*4]
lw   r2, [SP,2*4]
lw   r3, [SP,3*4]
lw   r4, [SP,4*4]
lw   r5, [SP,5*4]
lw   r6, [SP,6*4]
lw   r7, [SP,7*4]
lw   r8, [SP,8*4]
lw   r9, [SP,9*4]
lw   r10, [SP,10*4]
lw   r11, [SP,11*4]
lw   r12, [SP,12*4]
lw   r13, [SP,13*4]
lw   r14, [SP,14*4]
```

```
lw    r15, [SP,15*4]
lw    r16, [SP,16*4]
lw    r17, [SP,17*4]
lw    r18, [SP,18*4]
lw    r19, [SP,19*4]
lw    r20, [SP,20*4]
lw    r21, [SP,21*4]
lw    r22, [SP,22*4]
lw    r23, [SP,23*4]
lw    r24, [SP,24*4]
lw    r25, [SP,25*4]
lw    r26, [SP,26*4]
lw    r27, [SP,27*4]
lw    r28, [SP,28*4]
lw    r29, [SP,29*4]
lw    r30, [SP,30*4]
lw    r31, [SP,31*4]
lw    r30, [SP,33*4]
lw    r31, [SP,35*4]
mtcr        r30, cr1
mtcr        r31, cr5
addi SP, 38*4
rte
```

Sys_IRQ.c 定义了 irq_dispatch(cp0_cause)函数和 intmsg()函数：irq_dispatch(cp0_cause)函数在 Sys_isr.S 文件中的 save_reg 函数中调用，为中断的分支函数，cp0_cause 为 cr2 的内容，根据 cr2 的内容调用 IRQ63()~IRQ24()中的相应中断服务程序；intmsg()函数函数在 Sys_isr.S 文件中的 norm_debug_service 函数、general_service 函数和 int_service 函数中调用，为两个软件中断和除 SPCE3200 的 40 个硬件中断外的其他硬件中断的处理程序。程序段如下：

```
//=================================================================
//文 件 名：Sys_IRQ.c
//功能描述：SPCE3200 40 个中断源（硬件）的中断向量分支及软件中断处理。
```

```
//        使用时，用户不需要修改此文件中的内容
//===========================================================================


//===============================================================
// 语法格式：void intmsg(void)
// 功能描述：软件中断及除 SPCE3200 40 个硬件中断之外的
//           S+core 内核硬件中断处理程序
// 入口参数：无
// 出口参数：无
//===============================================================
void intmsg(void)
{
    while(1);
}

//===============================================================
// 语法格式：void irq_dispatch(unsigned int cp0_cause)
// 功能描述：取 SPCE3200 40 个硬件中断的中断向量号，
//           根据中断向量号调用相应中断服务程序
// 入口参数：cp0_cause 为 S+core 内核寄存器 cr2 的内容
//           其中第 23~18 位为 63 个硬件中断的中断请求状态（中断向量号）
// 出口参数：无
//===============================================================
void irq_dispatch(unsigned int cp0_cause)
{
    int intvec=0;

    intvec = (cp0_cause & 0x00FC0000)>>18; //  取中断向量号

    switch (intvec)
    {
        case 63:        // DAC 中断
```

```
                IRQ63(); // IRQ63()函数在 User_IRQ.c 文件中定义
                break;
            case 62:        // 保留
                IRQ62(); // IRQ62()函数在 User_IRQ.c 文件中定义
                break;
            ……
            case 56:        // Timer 中断
                IRQ56(); // IRQ56()函数在 User_IRQ.c 文件中定义
                break;
            ……
            case 25:        // 保留
                IRQ25(); // IRQ25()函数在 User_IRQ.c 文件中定义
                break;
            case 24:        // 保留
                IRQ24(); // IRQ24()函数在 User_IRQ.c 文件中定义
                break;
            default:
                break;
        }
        return;
    }
```

User_IRQ.c 中定义了 IRQ63()~IRQ24()四十个中断服务函数，为一个用户接口函数文件，用户可以在其中添加用户中断服务函数。

```
//=====================================================================
//文 件 名：User_IRQ.c
//功能描述：SPCE3200 40 个中断源的用户中断服务函数。
//          根据中断向量号，用户可编写相应中断服务函数
//=====================================================================
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
```

```
//=========================================================
// 语法格式：void IRQ63(void)
// 功能描述：DAC 中断服务函数
// 入口参数：无
// 出口参数：无
//=========================================================
void IRQ63(void)
{

}
//=========================================================
// 语法格式：void IRQ62(void)
// 功能描述：保留中断向量中断服务函数
// 入口参数：无
// 出口参数：无
//=========================================================
void IRQ62(void)
{

}
……
//=========================================================
// 语法格式：void IRQ56(void)
// 功能描述：定时器（Timer）中断服务函数
// 入口参数：无
// 出口参数：无
//=========================================================
void IRQ56(void)
{

}
……
```

```
//=====================================================
// 语法格式：void IRQ25(void)
// 功能描述：保留中断向量中断服务函数
// 入口参数：无
// 出口参数：无
//=====================================================
void IRQ25(void)
{

}
//=====================================================
// 语法格式：void IRQ24(void)
// 功能描述：保留中断向量中断服务函数
// 入口参数：无
// 出口参数：无
//=====================================================
void IRQ24(void)
{

}
```

在使用 SPCE3200 的 40 个中断（63~24）时，各函数之间的调用关系如图 3.7。当有中断请求时，先按照地址从 IntX_vec 入口，IntX_vec 为 Int63_vec~Int24_vec；在 IntX_vec 中调用 save_reg 函数进行中断服务的处理；在 save_reg 函数中先保存现场，取中断向量号并把中断向量号作为参数，调用 irq_dispatch 函数；save_reg 函数中把中断向量号作为参数传递给 irq_dispatch 函数，irq_dispatch 函数按照中断向量号调用相应的中断服务函数 IRQX()，IRQX 为 IRQ24()~IRQ63()。

所以 SPCE3200 的中断处理过程也可看作图 3.8 的处理过程：图 3.7 的调用关系为中断从响应入口到处理完中断服务程序的过程，即左边粗箭头的处理过程；处理中断服务程序后，按照右边粗箭头的过程恢复现场，完成一次中断的处理过程。

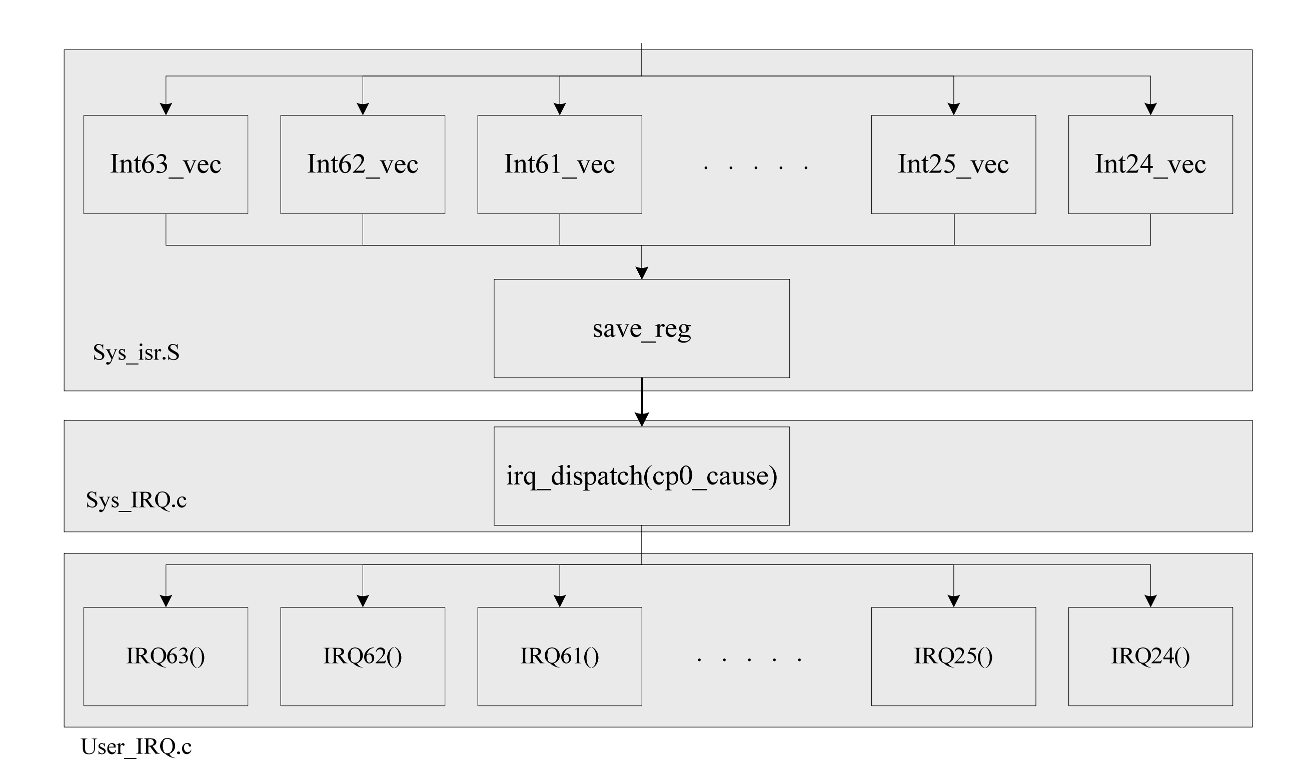


图 3.7 中断各函数的调用关系图

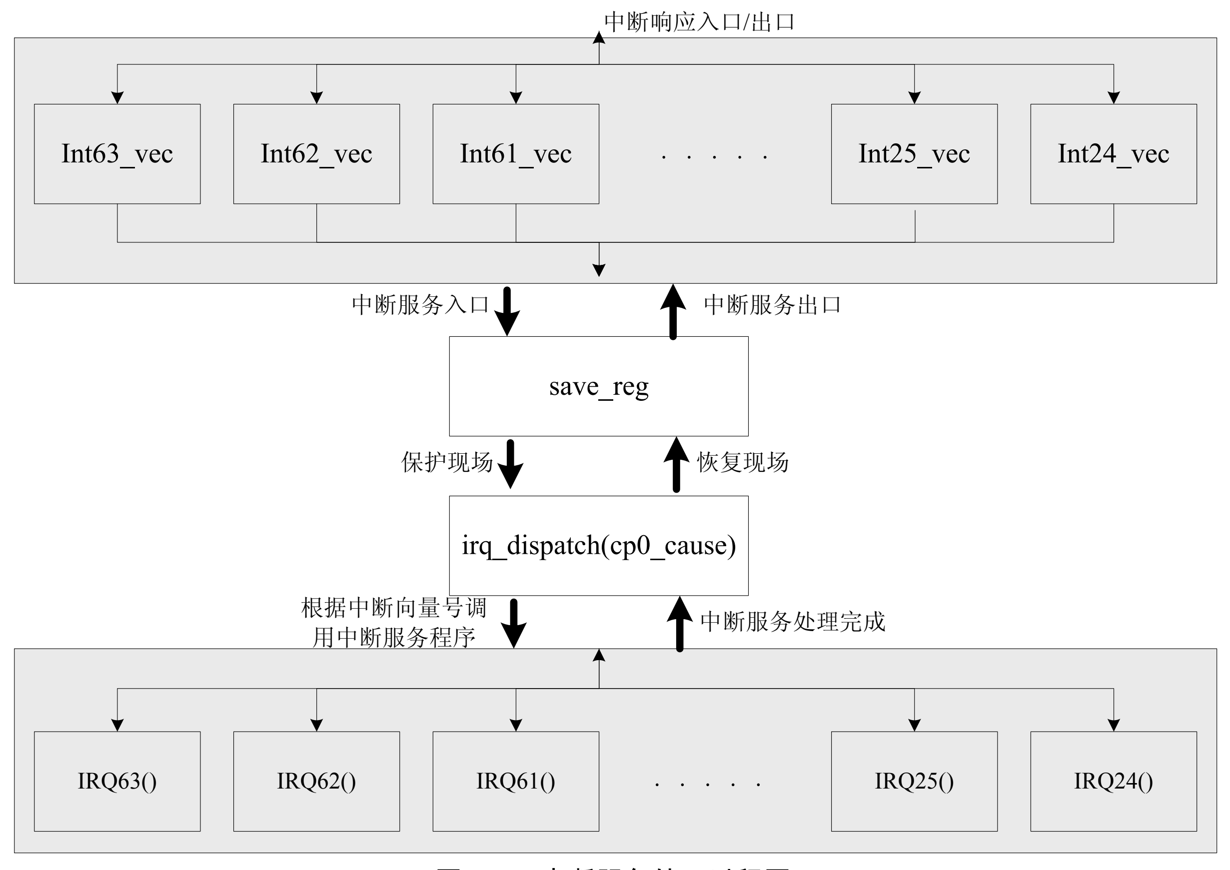


图 3.8 中断服务处理过程图

注意：如果没有特别说明，IRQ24()~IRQ63()中断服务函数中都需要手动清除中断标志。

3.6.7 应用举例

【例 3.1】

利用 Timer 中断实现三个发光二极管电平翻转（即闪烁）。

分析：根据图 3.6，中断控制器必须关闭来自 Timer 中断源的中断屏蔽；而且，必须通过相应寄存器使能 Timer 模块中断，Timer 才会发出中断请求；所以主程序中主要进行中断使能的操作。

按照表 4.52，Timer 中断源对应的中断向量号为 56，则根据 3.6.6 节中断机制可指导，中断服务处理中只需要在 IRQ56()函数中编写二极管电平翻转程序即可。主程序参考程序如下：

```
int main()
{
    * P_IOB_GPIO_SETUP = 0x00003800;
    *P_INT_MASK_CTRL1 = 0xffffff7f;                // 打开来自 Timer 中断源的中断

// *P_CLK_32K_CTRL = C_32K_CRY_EN;                 // 如果选择 32768Hz 时钟使能晶振
    *P_TIMER0_CLK_CONF = C_TIMER_CLK_EN
                        | C_TIMER_RST_DIS;         // 配置 Timer0 的时钟
    *P_CKG_SEL_TIMER = C_TIMER0_CLK_27M
                        | 0x8;                     // 设置定时器源时钟频率为 27MHZ/9
    *P_TIMER0_PRELOAD_DATA = 0x80;                 // 设置计数初值
    *P_TIMER0_CCP_CTRL = C_TIMER_NOR_MODE;// 工作模式选择定时计数模式
    *P_TIMER0_MODE_CTRL = C_TIMER_CTRL_EN
                        | C_TIMER_INT_EN
                        | C_TIMER_INT_FLAG;// 使能定时器、使能中断、清中断
    while(1);
}
```

中断服务参考程序如下：

```
void IRQ56(void)
{
    if(*P_TIMER0_MODE_CTRL & C_TIMER_INT_FLAG)
    {
        *P_TIMER0_MODE_CTRL |= C_TIMER_INT_FLAG;     //清除定时器中断
        *P_IOB_GPIO_SETUP ^= 0x00000038;             //LED1~3 反相，
```

```
        }
    }
```

3.7 存储器接口单元——MIU

SPCE3200 芯片上具备存储器接口单元（MIU——Memory Interface Unit），可以更好的设计存储器接口。结构如图 3.9 所示：

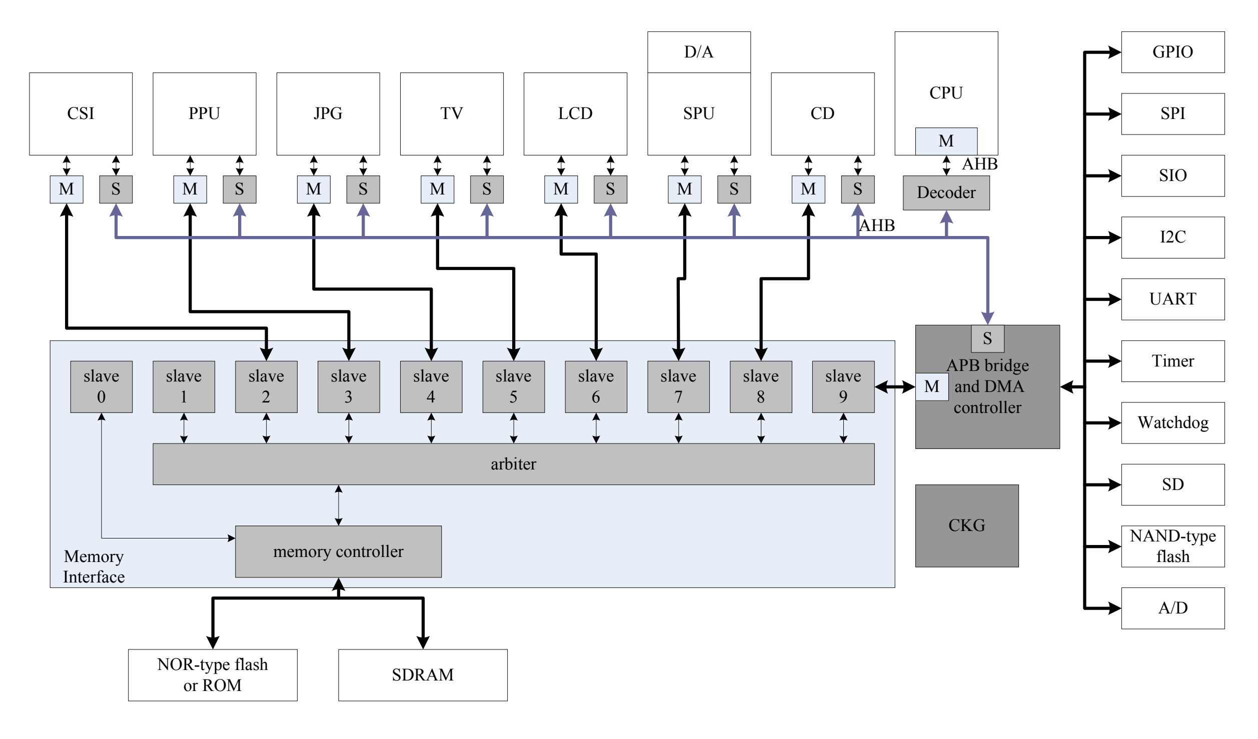


图 3.9 MIU 结构图

MIU 存储接口寄存器主要涉及 TV、LCD、SDRAM、MP4 模块，这里只介绍与 SDRAM 相关的寄存器，与其他模块相关寄存器只列出，更详细的介绍请参考相关章节。

表 3.27 MIU 涉及模块相关寄存器

TV 相关寄存器	P_TV_BUFFER_SA1
	P_TV_BUFFER_SA2
	P_TV_BUFFER_SA3
LCD 相关寄存器	P_LCD_BUFFER_SA1
	P_LCD_BUFFER_SA2
	P_LCD_BUFFER_SA3
MPEG4 相关寄存器	P_MPEG4_RAWBUFFER_SA1

	P_MPEG4_RAWBUFFER_SA2
	P_MPEG4_RAWBUFFER_SA3
	P_MPEG4_WRITEBUFFER_SA1
	P_MPEG4_WRITEBUFFER_SA2
	P_MPEG4_WRITEBUFFER_SA3
	P_MPEG4_VLCBUFFER_SA1
	P_MPEG4_VLCBUFFER_SA2
	P_MPEG4_FRAMEBUFFER_HSIZE
SDRAM 相关寄存器	P_MIU_SDRAM_POWER
	P_MIU_SDRAM_SETUP1
	P_MIU_SDRAM_SETUP2
	P_MIU_SDRAM_STATUS
MIU 时钟寄存器	P_MIU_CLK_CONF

MIU 时钟配置寄存器：P_MIU_CLK_CONF(0x88210010)

MIU 时钟配置寄存器用于使能或禁止 MIU 模块时钟，或软复位 MIU 模块。

表 3.28 P_MIU_CLK_CONF(0x88210010)

位	b31~b3	b2	b1	b0
读/写	-	R/W	R/W	R/W
默认值	-	0	0	0
名称	-	MIU_RST	MIU_DRAM	MIU_STOP

MIU_RST　　b2　　MIU 模块时钟复位位：

0：MIU 模块时钟复位

1：MIU 模块时钟不复位

MIU_DRAM　　b1　　强制 SDRAM 进入自刷新模式同时停止 MIU 时钟直到唤醒发生：

0：使能进入自刷新模式

1：不使能进入自刷新模式

MIU_STOP　　b0　　MIU 模块时钟停止位：

0：MIU 模块时钟停止

1：MIU 模块时钟使能

SDRAM 电源寄存器：P_MIU_SDRAM_POWER(0x8807005C)

SDRAM 电源设置。

表 3.29 P_MIU_SDRAM_POWER(0x8807005C)

位	b31~b4	b3	b2	b1	b0
读/写	-	R/W	R/W	R/W	R/W
默认值	-	0	0	0	0
名称	-	DEEP	REDO_MRS	POWER_DW	SELF_REF

DEEP　　b3　　强制 SDRAM 进入深度省电模式使能位：

0：不使能 SDRAM 进入深度省电模式

1：使能 SDRAM 进入深度省电模式

REDO_MRS　　b2　　重新设置模式寄存器使能位：

0：不使能重新设置模式寄存器

1：使能重新设置模式寄存器

POWER_DW　　b1　　强制 SDRAM 进入省电模式使能位：

0：不使能 SDRAM 进入省电模式

1：使能 SDRAM 进入省电模式

SELF_REF　　b0　　强制 SDRAM 进入自动刷新模式：

0：不使能 SDRAM 进入自动刷新模式

1：使能 SDRAM 进入自动刷新模式

SDRAM 参数设置寄存器：1 P_MIU_SDRAM_SETUP1(0x88070060)

SDRAM 的相关参数设置。

表 3.30 P_MIU_SDRAM_SETUP1(0x88070060)

位	b31	b30~b27	b26~b25	b24~b23	b22~b15
读/写	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0

名称	MIU_EN	SDCLK_SEL	SDRAM_RN	SDRAM_CW	SFRFR_PERIOD
位	**b14~b11**	**b10~b3**	**b2**	**b1**	**b0**
读/写	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0
名称	RFR_CYCLE_NUM	ATRFR_PERIOD	CAS_LA_CYCLE	tRCD_CYCLE	tRP_CYCLE

MIU_EN　　b31　　MIU 使能位：

0：不使能 MIU

1：使能 MIU

SDCLK_SEL　　b30~b27　　SDRAM 时钟选择位：

选择 SDRAM_CLK 的相位延迟

SDRAM_RN　　b26~b25　　每个逻辑 Bank（L-Bank）包含的行数选择位：

00：1024 行每逻辑 Bank

01：2048 行每逻辑 Bank

10：4096 行每逻辑 Bank

11：8192 行每逻辑 Bank

SDRAM_CW　　b24~b23　　SDRAM 每列包含的位数：

00：8 位宽度

01：16 位宽度

10：32 位宽度

11：保留

SFRFR_PERIOD　　b22~b15　　若在（SFRFR_PERIOD × 256 × ATRFR_PERIOD × 16）个时钟周期里没有发生 ADG_REQ，则会由硬件触发自动刷新动作

RFR_CYCLE_NUM　　b14~b11　　确定自动刷新需要的时钟周期数

ATRFR_PERIOD　　b10~b3　　每隔（ATRFR_PERIOD × 16）个时钟周期产生自动刷新动作

CAS_LA_CYCLE　　b2　　Cas latency 时钟周期设定

0：2 个时钟周期（默认设置）

1：3 个时钟周期

tRCD_CYCLE　　b1　　tRCD 时钟周期设定

0：1 个时钟周期

1：2 个时钟周期（默认设置）

tRP_CYCLE b0 tRP 时钟周期设定

0：1 个时钟周期

1：2 个时钟周期（默认设置）

SDRAM 参数设置寄存器 2： P_MIU_SDRAM_SETUP2(0x88070094)

SDRAM 的相关参数设置。

表 3.31 P_MIU_SDRAM_SETUP2(0x88070094)

位	b1	b0
读/写	R/W	R/W
默认值	0	0
名称	tRCD_CYCLE	tRP_CYCLE

tRCD_CYCLE b1 tRCD 时钟周期设置：

0：参考 P_MIU1_SDRAM_SETTING[1]位设置

1：3 个时钟周期

tRP_CYCLE b0 tRP 时钟周期设置

0：参考 P_MIU1_SDRAM_SETTING[0]位设置

1：3 个时钟周期

SDRAM 状态寄存器：P_MIU _SDRAM_STATUS(0x8807006C)

反映 MIU 的工作状态。

表 3.32 P_MIU_SDRAM_STATUS(0x8807006C)

位	b0
读/写	R/W
默认值	0
名称	MIU_STATUS

MIU_STATUS b0 SDRAM 的状态位：

0：SDRAM 没有状态发生

1：SDRAM 处于自动刷新模式或省电模式

3.8 APB 总线 DMA

SPCE3200 具有 DMA 功能，通过 DMA 功能可以完成 APB 总线读取 APB 外设模块数据写到 MIU 存储器，或从 MIU 存储器读取数据写到 APB 外设模块。APBDMA 结构如图 3.10 所示：

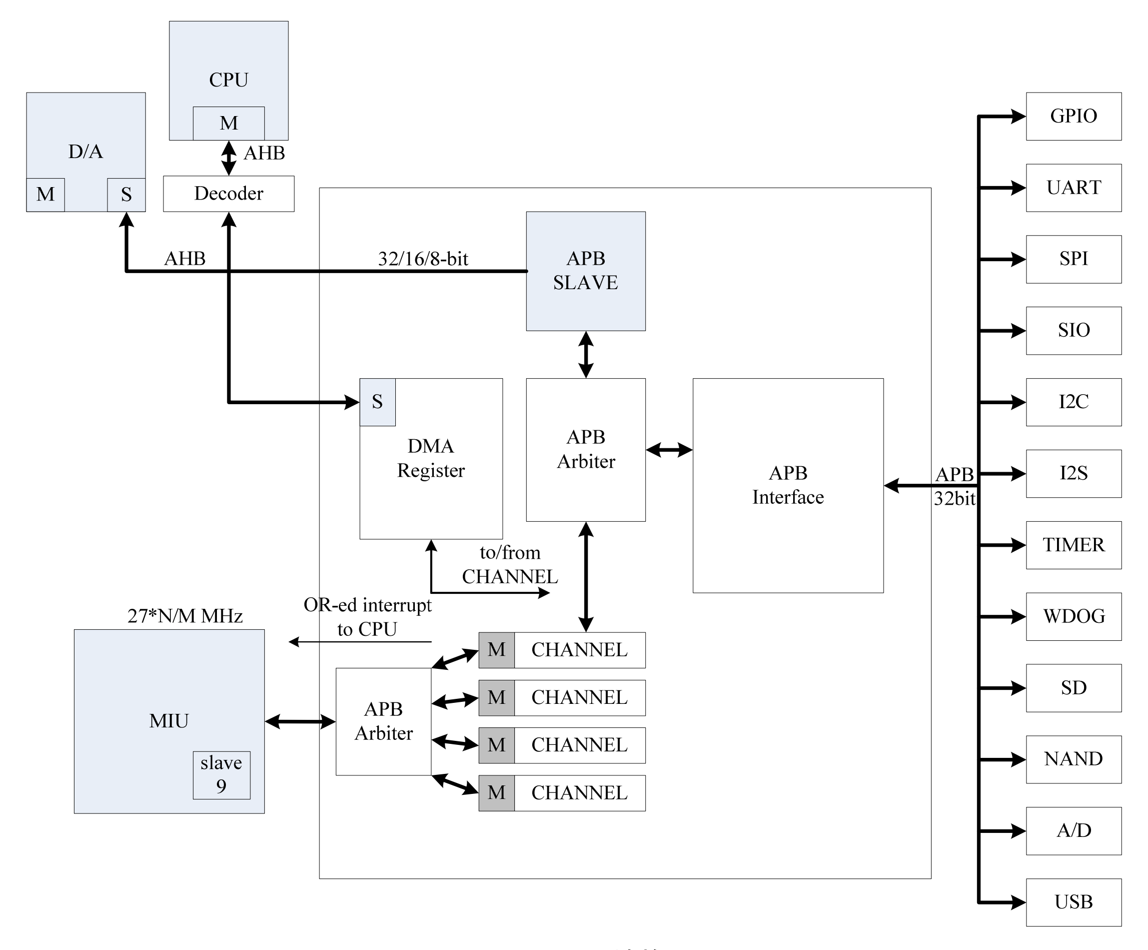


图 3.10 APBDMA 结构图

DMA 控制器可为 APB 外设模块同时提供 4 个用于读/写 MIU 存储器的通道，每一通道都可以设置成下列 4 类传输方式：

- 8 位单通道传输
- 16 位单通道传输
- 32 位单通道传输
- 32 位突发模式传输

DMA 具有 2 种启动方式：

1，当 DMA 通道被使能（通道使能位被设为 1）后，DMA 控制器通过通道连续地读出或写入数据，并在完成读写操作后结束 DMA 操作；

2，当 DMA 通道被使能后，在 APB 外设模块发出 REQ 请求时 DMA 控制器便会进行读写操作

一次，并当所有 REQ 请求的读写操作完成后结束操作；

当 DMA 控制器读/写 MIU 时有 2 种存储器存储方式：

1，单缓存区方式：即指定 DMA 缓存区的起始、结束地址，则 DMA 控制器仅对这一存储空间进行读写操作，并在读写操作完成时通过发出 IRQ 信号来结束操作；

2，双缓存区方式：即同时指定 BUFFER A 和 BUFFER B 缓存区的起始、结束地址，则 DMA 控制器会交替对 BUFFER A 和 BUFFER B 进行读写操作，并在二者有关读写操作的传输完成后发出 IRQ 信号。将 DMA 通道使能位设回到‘0’时，会产生 DMA 操作终止请求，则 DMA 控制器会在当前传输完成时结束其操作。

当 DMA 向 APB 外设模块进行读写操作时，有 2 种寻址方式来定位外设端口：

1，常规寻址；

2，连续寻址。

注意：当 DMA 操作完成后，用户需要写‘1’到相应的通道 IRQ 状态位以清除 DMA 中断，并需要写‘0’到相应的通道 DMA 使能位以结束其 DMA 操作。

DMA 相关寄存器：

DMA 控制器共有 28 个寄存器，如表 3.33 所示。通过对这 28 个寄存器的操作，即可使用 DMA 操作。下面将对这些寄存器一一进行说明。

表 3.33 DMA 控制器相关寄存器列表

寄存器名称	助记符	地址
DMA 时钟配置寄存器	P_DMA_CLK_CONF	0x88210058
DMA 忙状态寄存器	P_DMA_BUSY_STATUS	0x88080000
DMA 中断状态寄存器	P_DMA_INT_STATUS	0x88080004
AHB DMA 第 0 通道缓冲区 A 起始地址	P_DMA_AHB_SA0BA	0x88080008
AHB DMA 第 1 通道缓冲区 A 起始地址	P_DMA_AHB_SA1BA	0x8808000C
AHB DMA 第 2 通道缓冲区 A 起始地址	P_DMA_AHB_SA2BA	0x88080010
AHB DMA 第 3 通道缓冲区 A 起始地址	P_DMA_AHB_SA3BA	0x88080014
AHB DMA 第 0 通道缓冲区 A 结束地址	P_DMA_AHB_EA0BA	0x88080018
AHB DMA 第 1 通道缓冲区 A 结束地址	P_DMA_AHB_EA1BA	0x8808001C
AHB DMA 第 2 通道缓冲区 A 结束地址	P_DMA_AHB_EA2BA	0x88080020
AHB DMA 第 3 通道缓冲区 A 结束地址	P_DMA_AHB_EA3BA	0x88080024
APB DMA 第 0 通道起始地址	P_DMA_APB_SA0	0x88080028

寄存器名称	助记符	地址
APB DMA 第 1 通道起始地址	P_DMA_APB_SA1	0x8808002C
APB DMA 第 2 通道起始地址	P_DMA_APB_SA2	0x88080030
APB DMA 第 4 通道起始地址	P_DMA_APB_SA3	0x88080034
AHB DMA 第 0 通道缓冲区 B 起始地址	P_DMA_AHB_SA0BB	0x8808004C
AHB DMA 第 1 通道缓冲区 B 起始地址	P_DMA_AHB_SA1BB	0x88080050
AHB DMA 第 2 通道缓冲区 B 起始地址	P_DMA_AHB_SA2BB	0x88080054
AHB DMA 第 3 通道缓冲区 B 起始地址	P_DMA_AHB_SA3BB	0x88080058
AHB DMA 第 0 通道缓冲区 B 结束地址	P_DMA_AHB_EA0BB	0x8808005C
AHB DMA 第 1 通道缓冲区 B 结束地址	P_DMA_AHB_EA1BB	0x88080060
AHB DMA 第 2 通道缓冲区 B 结束地址	P_DMA_AHB_EA2BB	0x88080064
AHB DMA 第 3 通道缓冲区 B 结束地址	P_DMA_AHB_EA3BB	0x88080068
DMA 第 0 通道控制寄存器	P_DMA_CHANNEL0_CTRL	0x8808006C
DMA 第 1 通道控制寄存器	P_DMA_CHANNEL1_CTRL	0x88080070
DMA 第 2 通道控制寄存器	P_DMA_CHANNEL2_CTRL	0x88080074
DMA 第 3 通道控制寄存器	P_DMA_CHANNEL3_CTRL	0x88080078
DMA 通道复位寄存器	P_DMA_CHANNEL_RESET	0x8808007C

DMA 时钟配置寄存器：P_DMA_CLK_CONF(0x88210058)

DMA 时钟配置寄存器用于使能或禁止 DMA 控制器模块时钟，或软复位 DMA 控制器模块。

表 3.34 P_DMA_CLK_CONF(0x88210058)

位	b23~b21	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	APBDMA_RST	APBDMA_STOP

APBDMA_RST　　b1　　APBDMA 模块时钟复位位：

0：APBDMA 模块时钟复位

1：APBDMA 模块时钟不复位

APBDMA_STOP　　b0　　　　APBDMA 模块时钟停止位：

0：APBDMA 模块时钟停止

1：APBDMA 模块时钟使能

DMA 忙状态寄存器：P_DMA_BUSY_STATUS(0x88080000)

DMA 忙状态寄存器反映 DMA 的 4 个通道是否出于忙状态。

表 3.35 P_DMA_BUSY_STATUS(0x88080000)

位	b3	b2	b1	b0
读/写	R	R	R	R
默认值	0	0	0	0
名称	CH3_BUSY	CH2_BUSY	CH1_BUSY	CH0_BUSY

CH3_BUSY　　b3　　　　为 1 时表示通道 3 处于忙状态

CH2_BUSY　　b2　　　　为 1 时表示通道 2 处于忙状态

CH1_BUSY　　b1　　　　为 1 时表示通道 1 处于忙状态

CH0_BUSY　　b0　　　　为 1 时表示通道 0 处于忙状态

DMA 中断状态寄存器：P_DMA_INT_STATUS(0x88080004)

DMA 中断状态寄存器反映 DMA 的 4 个通道的中断是否发生、中断标志清除等操作。

表 3.36 P_DMA_INT_STATUS(0x88080004)

位	b3	b2	b1	b0
读/写	R/W	R/W	R/W	R/W
默认值	0	0	0	0
名称	CH3_IRQ	CH2_IRQ	CH1_IRQ	CH0_IRQ

CH3_IRQ　　b3　　　　DMA 通道 3 的 IRQ 中断标志位：

读 0：未发生 DMA IRQ 中断

读 1：发生 DMA IRQ 中断

写 0：无意义

写 1：清除 DMA IRQ 状态标志

CH2_IRQ b2 DMA 通道 3 的 IRQ 中断标志位：

读 0：未发生 DMA IRQ 中断

读 1：发生 DMA IRQ 中断

写 0：无意义

写 1：清除 DMA IRQ 状态标志

CH1_IRQ b1 DMA 通道 3 的 IRQ 中断标志位：

读 0：未发生 DMA IRQ 中断

读 1：发生 DMA IRQ 中断

写 0：无意义

写 1：清除 DMA IRQ 状态标志

CH0_IRQ b0 DMA 通道 3 的 IRQ 中断标志位：

读 0：未发生 DMA IRQ 中断

读 1：发生 DMA IRQ 中断

写 0：无意义

写 1：清除 DMA IRQ 状态标志

AHB DMA 第 0 通道缓存区 A 起始地址：P_DMA_AHB_SA0BA(0x88080008)

AHB DMA 第 1 通道缓存区 A 起始地址：P_DMA_AHB_SA1BA(0x8808000C)

AHB DMA 第 2 通道缓存区 A 起始地址：P_DMA_AHB_SA2BA(0x88080010)

AHB DMA 第 3 通道缓存区 A 起始地址：P_DMA_AHB_SA3BA(0x88080014)

AHB DMA 第*通道缓存区 A 起始地址寄存器设置第*通道缓存区 A 的起始地址。

表 3.37 P_DMA_AHB_SAxBA

寄存器名称	b31~b0
P_DMA_AHB_SA0BA	第 0 通道缓存区 A 起始地址
P_DMA_AHB_SA1BA	第 1 通道缓存区 A 起始地址
P_DMA_AHB_SA2BA	第 2 通道缓存区 A 起始地址
P_DMA_AHB_SA3BA	第 3 通道缓存区 A 起始地址

AHB DMA 第 0 通道缓存区 A 结束地址：P_DMA_AHB_EA0BA(0x88080018)

AHB DMA 第 1 通道缓存区 A 结束地址：P_DMA_AHB_EA1BA(0x8808001C)

AHB DMA 第 2 通道缓存区 A 结束地址：P_DMA_AHB_EA2BA(0x88080020)

AHB DMA 第 3 通道缓存区 A 结束地址：P_DMA_AHB_EA3BA(0x88080024)

AHB DMA 第*通道缓存区 A 结束地址寄存器设置第*通道缓存区 A 的结束地址。

表 3.38 P_DMA_AHB_EAxBA

寄存器名称	b31~b0
P_DMA_AHB_EA0BA	第 0 通道缓存区 A 结束地址
P_DMA_AHB_EA1BA	第 1 通道缓存区 A 结束地址
P_DMA_AHB_EA2BA	第 2 通道缓存区 A 结束地址
P_DMA_AHB_EA3BA	第 3 通道缓存区 A 结束地址

APB DMA 第 0 通道起始地址：P_DMA_APB_SA0(0x88080028)

APB DMA 第 1 通道起始地址：P_DMA_APB_SA1(0x8808002C)

APB DMA 第 2 通道起始地址：P_DMA_APB_SA2(0x88080030)

APB DMA 第 3 通道起始地址：P_DMA_APB_SA3(0x88080034)

APB DMA 第*通道起始地址寄存器设置第*通道的起始地址。

表 3.39 P_DMA_APB_SAx

寄存器名称	b31~b0
P_DMA_APB_SA0	第 0 通道 APB 模块访问端口起始地址
P_DMA_APB_SA1	第 1 通道 APB 模块访问端口起始地址
P_DMA_APB_SA2	第 2 通道 APB 模块访问端口起始地址
P_DMA_APB_SA3	第 3 通道 APB 模块访问端口起始地址

AHB DMA 第 0 通道缓存区 B 起始地址：P_DMA_AHB_SA0BB(0x8808004C)

AHB DMA 第 1 通道缓存区 B 起始地址：P_DMA_AHB_SA1BB(0x88080050)

AHB DMA 第 2 通道缓存区 B 起始地址：P_DMA_AHB_SA2BB(0x88080054)

AHB DMA 第 3 通道缓存区 B 起始地址：P_DMA_AHB_SA3BB(0x88080058)

AHB DMA 第*通道缓存区 B 起始地址寄存器设置第*通道缓存区 B 的起始地址。

表 3.40 P_DMA_AHB_SAxBB

寄存器名称	b31~b0

P_DMA_AHB_SA0BB	第 0 通道缓存区 B 起始地址
P_DMA_AHB_SA1BB	第 1 通道缓存区 B 起始地址
P_DMA_AHB_SA2BB	第 2 通道缓存区 B 起始地址
P_DMA_AHB_SA3BB	第 3 通道缓存区 B 起始地址

AHB DMA 第 0 通道缓存区 B 结束地址：P_DMA_AHB_EA0BB(0x8808005C)

AHB DMA 第 1 通道缓存区 B 结束地址：P_DMA_AHB_EA1BB(0x88080060)

AHB DMA 第 2 通道缓存区 B 结束地址：P_DMA_AHB_EA2BB(0x88080064)

AHB DMA 第 3 通道缓存区 B 结束地址：P_DMA_AHB_EA3BB(0x88080068)

AHB DMA 第*通道缓存区 B 结束地址寄存器设置第*通道缓存区 B 的结束地址。

表 3.41 P_DMA_AHB_EAxBB

寄存器名称	b31~b0
P_DMA_AHB_EA0BB	第 0 通道缓存区 B 结束地址
P_DMA_AHB_EA1BB	第 1 通道缓存区 B 结束地址
P_DMA_AHB_EA2BB	第 2 通道缓存区 B 结束地址
P_DMA_AHB_EA3BB	第 3 通道缓存区 B 结束地址

DMA 第 0 通道控制寄存器：P_DMA_CHANNEL0_CTRL(0x8808006C)

DMA 第 1 通道控制寄存器：P_DMA_CHANNEL1_CTRL(0x88080070)

DMA 第 2 通道控制寄存器：P_DMA_CHANNEL2_CTRL(0x88080074)

DMA 第 3 通道控制寄存器：P_DMA_CHANNEL3_CTRL(0x88080078)

DMA 第*通道控制寄存器设置 DMA 控制器的使能、中断、传输方式选择位、存储器存储方式选择位、APB 外设模块定位寻址模式和读写方向等。

表 3.42 P_DMA_CHANNELx_CTRL

位	b7	b6	b5~b4	b3	b2	b1	b0
读/写	W	W	W	W	W	W	W
默认值	0	0	0	0	0	0	0
名称	CHx_EN	CHx_IRQ	CHx_TRANS	CHx_MEM	CHx_MODE	CHx_DMA	CHx_DIR

CHx_EN	b7	通道 x DMA 使能位： 0：禁止 1：使能
CHx_IRQ	b6	通道 x DMA 中断屏蔽： 0：屏蔽 DMA IRQ 1：允许 DMA IRQ
CHx_TRANS	b5~b4	通道 x 传输方式选择位： 00：8 位单通道传输方式 01：16 位单通道传输方式 10：32 位单通道传输方式 11：32 位突发模式传输
CHx_MEM	b3	通道 x MIU 存储器存储方式选择位： 0：单缓存区存储方式 1：双缓存区存储方式
CHx_MODE	b2	通道 x DMA 的 APB 外设模块定位寻址模式： 0：连续模式 1：常规模式
CHx_DMA	b1	通道 x DMA 方式选择位： 0：自动方式 1：查询方式
CHx_DIR	b0	通道 x 读写方向设置位： 0：MIU 向 APB 的读写 1：APB 向 MIU 的读写

DMA 通道复位寄存器：P_DMA_CHANNEL_RESET(0x8808007C)

DMA 通道复位寄存器复位 DMA 的通道。

表 3.43 P_DMA_CHANNEL_RESET(0x8808007C)

位	b3	b2	b1	b0
读/写	W	W	W	W
默认值	0	0	0	0

名称	CH3_REST	CH2_REST	CH1_REST	CH0_REST

CH3_REST　　b3　　通道 3 软件复位位：

0：不复位

1：复位

CH2_REST　　b2　　通道 2 软件复位位：

0：不复位

1：复位

CH1_REST　　b1　　通道 1 软件复位位：

0：不复位

1：复位

CH0_REST　　b0　　通道 0 软件复位位：

0：不复位

1：复位

3.9 启动代码

一般在 32 位开发平台的应用程序中，绝大部分都采用 C 语言编写，这样可以大大提高开发效率、增加软件可读性和软件可维护性。在系统初始化，通常会用一个汇编文件作为启动代码。

3.9.1 文件组成

启动文件由多个文件组成，包括 3.6.6 节讲的中断文件（当工程中要用到中断时）、*_startup.s 文件和*.ld 格式的文件，如图 3.11。

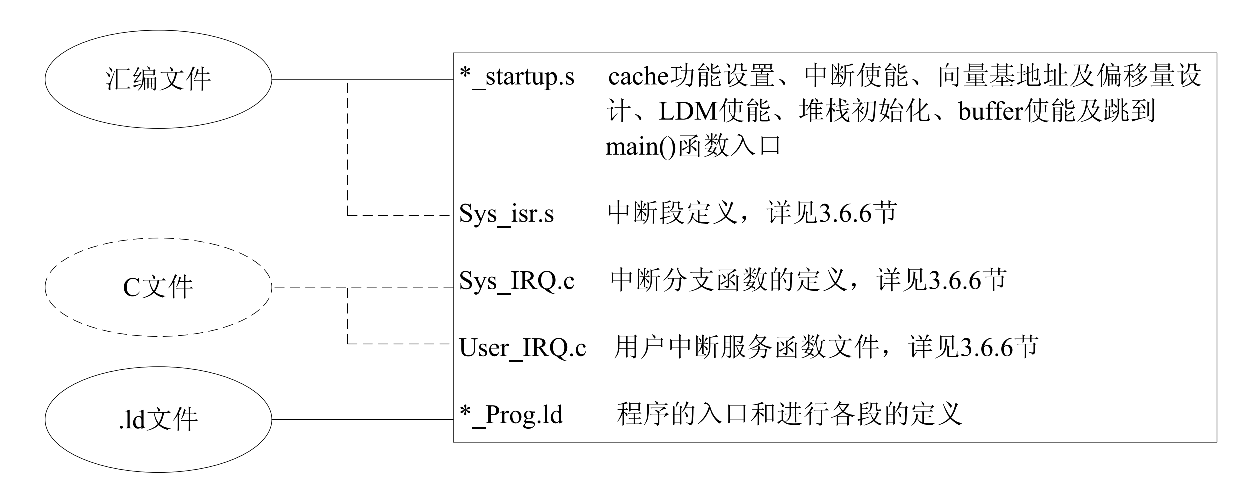


图 3.11　启动文件的组成

其中虚线指向的文件只有在中断才用到，可以看作是中断的启动文件，在 3.6.6 节已详细讲述，不再赘述。所以，对于一般的工程来说，整个启动代码由一个汇编文件（*_startup.s）和一个.ld 文件（*_Prog.ld）组成，“*”为工程名称。

3.9.2 *_Prog.ld

*_Prog.ld 文件主要有两大部分的内容：程序入口的定义和指定参加链接的段及段地址。

程序入口定义主要是指定程序运行的入口地址，利用 ENTRY()的格式来指定，括号内指定入口的标号。*_Prog.ld 文件中入口指定如下：

```
ENTRY(_hardware_init)                    //指定程序从_hardware_init 标号处入口
                                         //_hardware_init 在*_startup.s 文件中定义
```

指定参加链接的段包括.text、.data、.bss 等外，还包括用户定义的段（例如.hardware_init 段、.exception_vec 段等），如果用户自己利用.section 定义一个段，要在*_Prog.ld 文件中声明和控制分配地址。以下是两个常用的段的声明及地址的定义。

堆栈段：

```
.stack          0xa0fffffc :{ _stack = .; *(.stack) }     //定义堆栈段基地址为 0xa0fffffc
```

中断异常段：

```
.exception 0xA00001fc :                  //中断异常段，基地址为 0xA00001fc
{
    *(.exception_vec)                    //中断段名为.exception_vec，在
} = 0
```

3.9.3 *_startup.s

*_startup.s 文件首先定义了一个段.hardware_init（该段也在*_Prog.ld 文件中进行了链接声明），在.hardware_init 段中定义了_hardware_init 函数，且声明_hardware_init 函数为程序的入口。_hardware_init 函数里禁能了 cache 的写回功能；使能了 S+core 内核的 63 个硬件中断和 2 个软件中断，同时指定了中断向量基地址和偏移量；使能 LDM，初始化堆栈，启动 D-Cache 写缓存功能后进入 main 函数执行用户程序。

程序模块见图 3.12。

cache功能设置
中断使能及向量基地址和偏移量设置
LDM设置
堆栈初始化
BIU设置
跳转到main用户函数

图 3.12 *_startup.s 文件的程序功能模块示意图

程序段如下：

```
.extern _start
.section .hardware_init,"ax",@progbits        //定义.hardware_init 段

.global _hardware_init
//定义_hardware_init 函数作为程序启动入口，从*.ld 文件中 ENTRY 进入到_hardware_init 函数
.ent   _hardware_init
_hardware_init:
//========================================
// 禁能 cache 的数据写回
//========================================
    mfcr r5, cr4           //取出 Cache 控制寄存器 cr4 的数据存在 r5 中
    nop                    //
    li r7, 0x80            //把 0x80 装载到 r7 寄存器

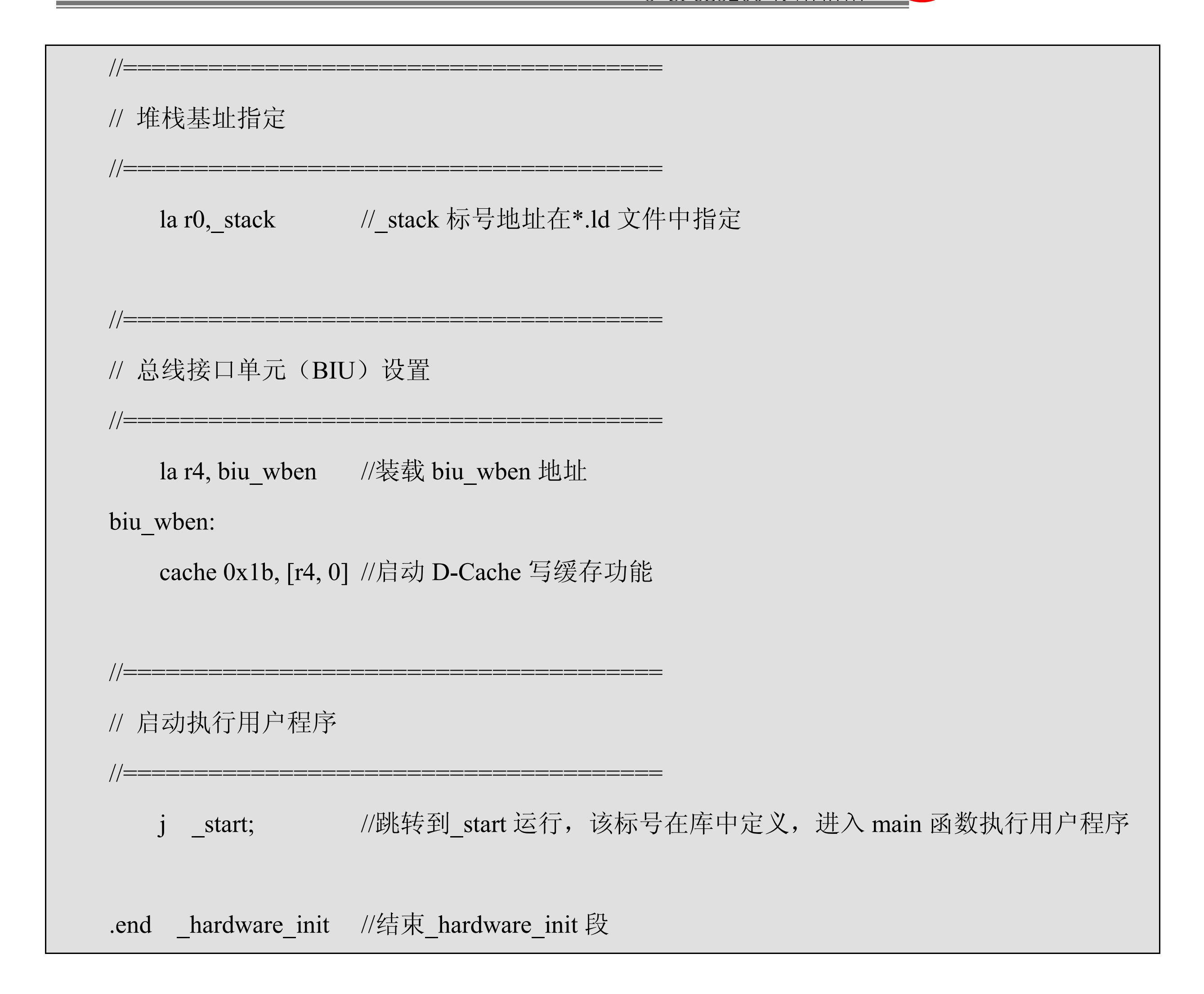
    andri r6, r5, 0x80     //取 r5（cr4）的 b7（cache 的写回使能）位，其他位置 0，存在 r6 中

    cmp.c r7, r6           //Cache 控制寄存器 cr4 的写回模式是否使能
    bne under_wt           //没有使能，跳到 under_wt
    nop
under_wb:                  //使能
```

```
        la r7, tgl_wb           //将 tgl_wb 的地址装载在 r7 中
        cache 0x1f, [r7, 0]
        //强制将一个被改写过的有效Cache行中内容写回主存储器并将全部D-Cache行改写为无效
        nop
        nop
        nop
tgl_wb:
        cache 0x1d, [r7, 0]  //禁止写回 D-Cache 功能
        nop
under_wt:                       //处理器进入写透模式

//==========================================
// 中断使能
//==========================================
        li r4, 0x1
        mtcr r4, cr0            //使能所有 63 个硬件中断和 2 个软件中断
        li r4, 0xa0000000
        mtcr r4, cr3
        //指定异常向量基地址为 0xa0000000,并指定向量地址的偏移模式为偏移 0x4

//==========================================
// LDM 使能
//==========================================
        li r5, 0xa1000000     //
        cache 0xb, [r5,0]     //指定 LDM 映射的起始地址
        mfcr r11, cr4
        ori r11, 0x8            //允许 LDM 接口
        bitset.c r11,r11,16   //使能省电模式
        mtcr r11, cr4
        nop
        nop
```



```
//========================================
// 堆栈基址指定
//========================================
        la r0,_stack            //_stack 标号地址在*.ld 文件中指定


//========================================
// 总线接口单元（BIU）设置
//========================================
        la r4, biu_wben         //装载 biu_wben 地址
biu_wben:
        cache 0x1b, [r4, 0]  //启动 D-Cache 写缓存功能


//========================================
// 启动执行用户程序
//========================================
        j   _start;             //跳转到_start 运行，该标号在库中定义，进入 main 函数执行用户程序

.end    _hardware_init     //结束_hardware_init 段
```

3.9.4 启动代码工作流程

SPCE3200 芯片在复位后，PC 寄存器指向 0x9F00 0000 地址，然后开始从 ENTRY 入口运行_hardware_init，对 D-cache、LDM、异常、堆栈等模块进行初始化，初始化完成后进入 main 函数执行用户程序。启动代码工作流程如图 3.13。

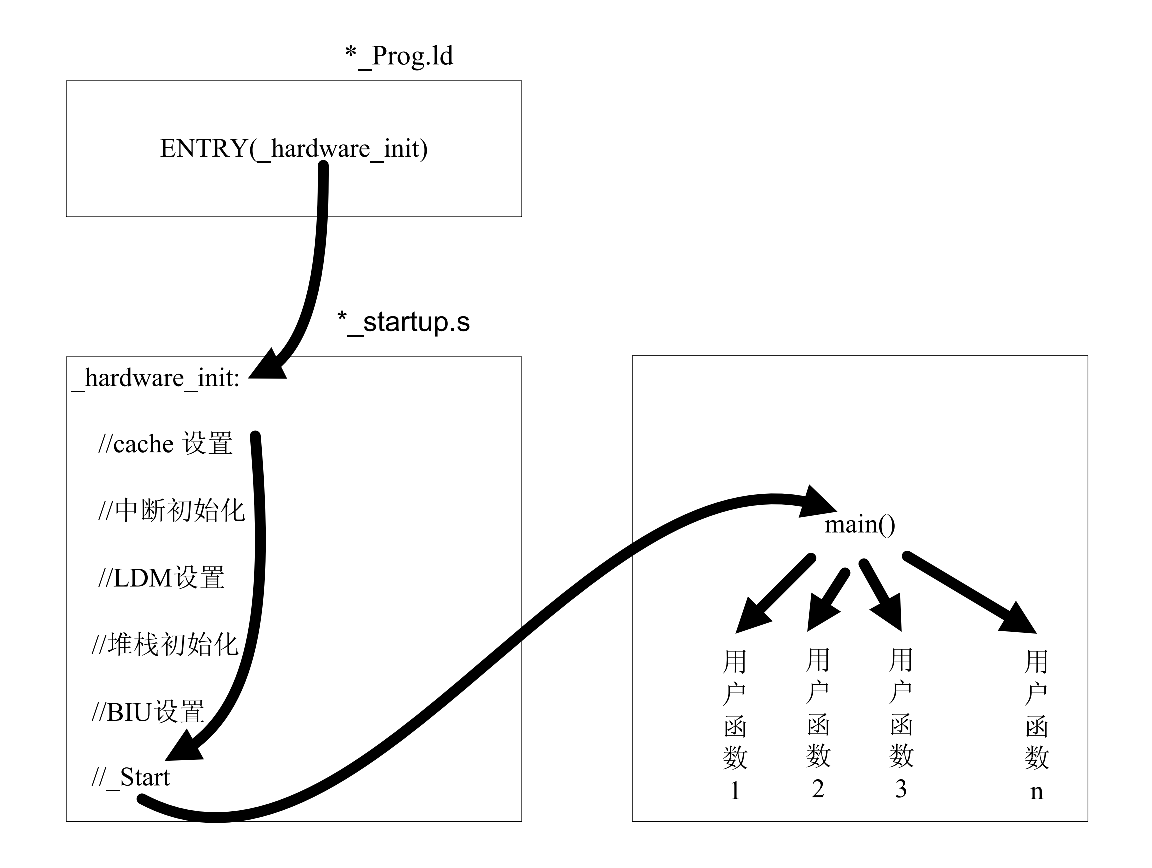


图 3.13　启动代码工作流程图



4 SPCE3200 功能部件

4.1 通用 I/O 口——GPIO

4.1.1 概述

SPCE3200 最多拥有 79 个通用 I/O 口（GPIO——General Purpose I/O ports），其中有 8 个 I/O 口只做为 GPIO 使用，其它 I/O 口是从其它模块复用为 GPIO 的，各个模块复用为 GPIO 口的情况参考表 4.1：

表 4.1 各个模块复用为 GPIO 情况

模块（module）	可以复用为 GPIO 的数目
TFT	20
CSI	13
NFLASH	16
JTAG	5
DRAM	4
USB	1
UART	2
IIC	2
ADC	8

每个 GPIO 可以单独编程设置为上拉电阻输入、下拉电阻输入、输出等。SPCE3200 中的部分 GPIO 可以作为外部中断端口，参考表 4.2：

表 4.2 GPIO 作为外部中断

模块（module）	可以作为外部中断的数目
TFT	4
CSI	4
NFLASH	4
JTAG	5
UART	2
IIC	2
ADC	4
USB	1

SPCE3200 的 8 个专门用作 GPIO 的端口分为 2 组，一组叫做 IOA 口，有 IOA0、IOA1 两个端口，可以作为外部中断；另一组叫做 IOB 口，有 IOB0、IOB1、IOB2、IOB3、IOB4、IOB5 六个端口，不可以作为外部中断。IOA 口与 IOB 口都可以编程设置为上拉输入、下拉输入、输出口等。本节主要介绍 IOA 口与 IOB 口，其它模块复用的 GPIO 参考相应模块章节有详细介绍。表 4.3 为可以作为 GPIO 所有端口的信息列表：

表 4.3 GPIO 信息表

序号	模块（module）	引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
1	TFT	LCD_CLK	148	bit[0]	可以
2		LCD_VS	147	bit[1]	可以
3		LCD_HS	146	bit[2]	可以
4		LCD_D0	144	bit[3]	可以
5		LCD_D1	143	bit[4]	不可以
6		LCD_D2	142	bit[5]	不可以
7		LCD_D3	141	bit[6]	不可以
8		LCD_D4	140	bit[7]	不可以
9		LCD_D5	139	bit[8]	不可以
10		LCD_D6	138	bit[9]	不可以
11		LCD_D7	137	bit[10]	不可以
12		LCD_D8	136	bit[11]	不可以
13		LCD_D9	135	bit[12]	不可以
14		LCD_D10	134	bit[13]	不可以
15		LCD_D11	133	bit[14]	不可以
16		LCD_D12	132	bit[15]	不可以
17		LCD_D13	131	bit[16]	不可以
18		LCD_D14	130	bit[17]	不可以
19		LCD_D15	129	bit[18]	不可以
20		LCD_ACT	145	bit[19]	不可以
21	CSI	CSI_D0	16	bit[0]	可以
22		CSI_D1	17	bit[1]	可以
23		CSI_D2	18	bit[2]	可以
24		CSI_D3	19	bit[3]	可以
25		CSI_D4	20	bit[4]	不可以



序号	模块（module）	引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
26		CSI_D5	21	bit[5]	不可以
27		CSI_D6	24	bit[6]	不可以
28		CSI_D7	25	bit[7]	不可以
29		CSI_CKO	26	bit[8]	不可以
30		CSI_CKI	27	bit[9]	不可以
31		CSI_HS	28	bit[10]	不可以
32		CSI_VS	29	bit[11]	不可以
33		CSI_FIELD	30	bit[12]	不可以
34	NFLASH	NF_ALE	127	bit[0]	可以
35		NF_WPN	126	bit[1]	可以
36		NF_CLE	125	bit[2]	可以
37		NF_REN	124	bit[3]	可以
38		NF_WEN	123	bit[4]	不可以
39		NF_CEN	119	bit[5]	不可以
40		NF_RDY	118	bit[6]	不可以
41		NF_D0	117	bit[7]	不可以
42		NF_D1	116	bit[8]	不可以
43		NF_D2	115	bit[9]	不可以
44		NF_D3	112	bit[10]	不可以
45		NF_D4	111	bit[11]	不可以
46		NF_D5	110	bit[12]	不可以
47		NF_D6	109	bit[13]	不可以
48		NF_D7	108	bit[14]	不可以
49		SPI_CSN	128	bit[15]	不可以
50	JTAG	JTAG_TRSTN	153	bit[0]	可以
51		JTAG_TDI	152	bit[1]	可以
52		JTAG_TDO	151	bit[2]	可以
53		JTAG_TCK	150	bit[3]	可以
54		JTAG_TMS	149	bit[4]	可以

序号	模块（module）	引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
55	DRAM	ROMCSN	15	bit[0]	不可以
56		DRAM_A11	239	bit[1]	不可以
57		DRAM_A12	14	bit[2]	不可以
58		DRAM_BA1	254	bit[3]	不可以
59	USB	USB_DET	156	bit[0]	可以
60	UART	UART_RX	40	bit[0]	可以
61		UART_TX	39	bit[1]	可以
62	IIC	IIC_CLK	35	bit[0]	可以
63		IIC_DATA	36	bit[1]	可以
64	ADC	ADC_CH0	160	bit[0]	可以
65		ADC_CH1	161	bit[1]	可以
66		ADC_CH2	162	bit[2]	可以
67		ADC_CH3	163	bit[3]	可以
68		ADC_CH4	164	bit[4]	不可以
69		ADC_CH5	165	bit[5]	不可以
70		ADC_CH6	166	bit[6]	不可以
71		ADC_CH7	167	bit[7]	不可以
72	IOA	IOA0	41	bit[0]	可以
73		IOA1	42	bit[1]	可以
74	IOB	IOB0	101	bit[0]	不可以
75		IOB1	100	bit[1]	不可以
76		IOB2	99	bit[2]	不可以
77		IOB3	98	bit[3]	不可以
78		IOB4	97	bit[4]	不可以
79		IOB5	96	bit[5]	不可以

4.1.2　引脚描述

SPCE3200 的 8 个 GPIO 引脚描述如表 4.4 所示：

表 4.4 GPIO 引脚描述

引脚名称	引脚号	引脚属性	引脚功能
IOA0	41	I/O	GPIO
IOA1	42	I/O	GPIO
IOB5	96	I/O	GPIO
IOB4	97	I/O	GPIO
IOB3	98	I/O	GPIO
IOB2	99	I/O	GPIO
IOB1	100	I/O	GPIO
IOB0	101	I/O	GPIO

4.1.3 结构

SPCE3200 的 IOA 口的结构如图 4.1 所示：可以通过寄存器 P_IOA_GPIO_SETUP 设置 IOA 口的上拉输入、下拉输入或输出口；通过寄存器 P_IOA_GPIO_INPUT 读入端口的值；通过设置 P_IOA_GPIO_INT 设置 IOA 口作为外部中断。

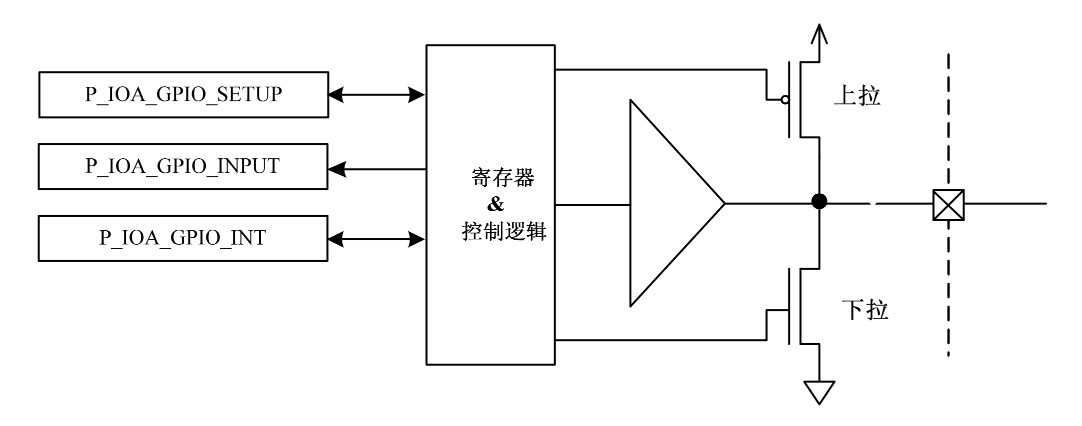


图 4.1 IOA 的结构图

SPCE3200 的 IOB 口的结构图如图 4.2 所示：可以通过寄存器 P_IOB_GPIO_SETUP 设置 IOB 口的上拉输入、下拉输入或输出口；通过寄存器 P_IOB_GPIO_INPUT 读出端口的值。

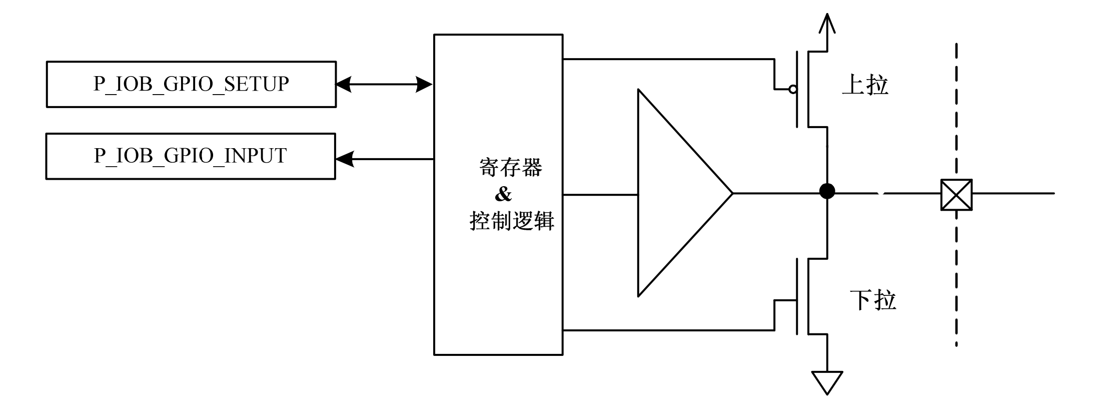


图 4.2 IOB 的结构图

4.1.4 寄存器描述

SPCE3200 的 8 个 GPIO 相关的寄存器有 6 个，参考表 4.5：

表 4.5 GPIO 相关寄存器描述

寄存器名称	助记符	地址
GPIO 时钟配置寄存器	P_GPIO_CLK_CONF	0x882100FC
IOA 设置寄存器	P_IOA_GPIO_SETUP	0x88200038
IOA 输入数据寄存器	P_IOA_GPIO_INPUT	0x88200074
IOA 口外部中断寄存器	P_IOA_GPIO_INT	0x88200090
IOB 口设置寄存器	P_IOB_GPIO_SETUP	0x8820004C
IOB 口输入数据寄存器	P_IOB_GPIO_INPUT	0x88200070

GPIO 时钟配置寄存器：P_GPIO_CLK_CONF(0x882100FC)

设置该寄存器可以使能 GPIO 模块，包括其他模块复用为 GPIO 时的情况，该寄存器默认就是使能的。

表 4.6 P_GPIO_CLK_CONF(0x882100FC)

位	b31~b26	b1	b0
读/写	-	R/W	R/W
默认值	-	1	1
名称	-	SFTCFG_RST	SFTCFG_STOP

SFTCFG_RST　b1　GPIO 模块时钟复位位：
0: GPIO 模块时钟复位
1: GPIO 模块时钟不复位

SFTCFG_STOP　b0　GPIO 模块时钟使能位：
0：GPIO 模块时钟停止
1：GPIO 模块时钟使能

IOA 设置寄存器：P_IOA_GPIO_SETUP(0x88200038)

IOA0、IOA1 的端口设置，当需要设置 IOA0、IOA1 为上拉电阻输入、下拉电阻输入或输出使能时需要设置该寄存器，当设置 IOA0、IOA1 为输出使能时，可以输出高电平或低电平。注意 IOA 的上拉电阻输入状态 0 为使能状态，1 为不使能状态。默认为下拉电阻输入状态。

表 4.7 P_IOA_GPIO_SETUP(0x88200038)

位	b31~b26	b25~b24	b23~b18	b17~b16	b15~b10	b9~b8	b7~b2	b1~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W
默认值	-	1	-	1	-	0	-	0
名称	-	IOA_PD	-	IOA_PU	-	IOA_OE	-	IOA_O

IOA1_PD　　b25　　IOA1 的下拉电阻输入使能位：
　　0: 不使能下拉电阻输入
　　1: 使能下拉电阻输入

IOA0_PD　　b24　　IOA0 的下拉电阻输入使能位：
　　0：不使能下拉电阻输入
　　1：使能下拉电阻输入

IOA1_PU　　b17　　IOA1 的上拉电阻输入使能位：
　　0：使能上拉电阻输入
　　1：不使能上拉电阻输入

IOA0_PU　　b16　　IOA0 的上拉电阻输入使能位：
　　0：使能上拉电阻输入
　　1：不使能上拉电阻输入

IOA1_OE　　b9　　IOA1 的输出使能位：
　　0：不使能输出
　　1：使能输出

IOA0_OE　　b8　　IOA0 的输出使能位：
　　0：不使能输出
　　1：使能输出

IOA1_O　　b1　　IOA1 的输出数据

IOA0_O　　b0　　IOA0 的输出数据

IOA 输入数据寄存器：P_IOA_GPIO_INPUT(0x88200074)

读该寄存器可以得到 IOA0~IOA1 的端口输入数据。

表 4.8 P_IOA_GPIO_INPUT(0x88200074)

位	b31~b2	b1~b0
读/写	-	R/W
默认值	-	0
名称	-	IOA_INPUT

GLOBAL_INPUT　b1~b0　　IOA1~IOA0 的端口输入数据

IOA 外部中断寄存器：P_IOA_GPIO_INT(0x88200090)

该寄存器为 IOA0~IOA1 的外部中断设置寄存器。

表 4.9 P_IOA_GPIO_INT(0x88200090)

位	b31~b26	b25~b24	b23~b18	b17~b16	b15~b10	b9~b8	b7~b2	b1~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W
默认值	-	0	-	0	-	0	-	0
名称	-	IOA_FI	-	IOA_RI	-	IOA_FIEN	-	IOA_RIEN

IOA_FI　　b25~b24　　IOA1~IOA0 端口下降沿中断标志位：

读 0：没有发生下降沿中断

读 1：发生下降沿中断

写 0：无意义

写 1：清除中断标志

IOA_RI　　b17~b16　　IOA1~IOA0 端口上升沿中断标志位：

读 0：没有发生上升沿中断

读 1：发生了上升沿中断

写 0：无意义

写 1：清除中断标志

IOA_FIEN　　b9~b8　　IOA1~IOA0 端口下降沿中断使能位：

0：不使能下降沿中断

1：使能下降沿中断

IOA_RIEN　　b1~b0　　IOA1~IOA0 端口上升沿中断使能位：

0：不使能上升沿中断

1：使能上升沿中断

IOB 设置寄存器：P_IOB_GPIO_SETUP(0x8820004C)

IOB0~IOB5 的端口设置，当需要设置 IOB0~IOB5 为上拉电阻输入、下拉电阻输入或输出使能时需要设置该寄存器，当设置 IOB0~IOB5 为输出使能时，可以输出高电平或低电平。

表 4.10 P_IOB_GPIO_SETUP(0x8820004C)

位	b31~b30	b29~b24	b23~b22	b21~b16	b15~b14	b13~b8	b7~b6	b5~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W
默认值	-	1	-	1	-	0	-	0

名称	-	IOB_PD	-	IOB_PU	-	IOB_OE	-	IOB_O

IOB_PD b29~b24 IOB5~IOB0 的下拉电阻输入使能位：
0：不使能下拉电阻输入
1：使能下拉电阻输入

IOB_PU b21~b16 IOB5~IOB0 的上拉电阻输入使能位：
0：使能上拉电阻输入
1：不使能上拉电阻输入

IOB_OE b13~b8 IOB5~IOB0 的输出使能位：
0：不使能输出
1：使能输出

IOB_O b5~b0 IOB5~IOB0 的输出数据

IOB 输入数据寄存器：P_IOB_GPIO_INPUT(0x88200070)

读该寄存器可以得到 IOB0~IOB5 的输入数据。

表 4.11 P_IOB_GPIO_INPUT(0x88200070)

位	b31~b14	b13~b8	b7~b0
读/写	-	R/W	-
默认值	-	0	-
名称	-	IOB_INPUT	-

IOB_INPUT b13~b8 IOB5~IOB0 的输入数据

4.1.5 基本操作

设置 IOA0、IOA1 为输入口，读取 IOA0、IOA1 的输入数据，参考程序段如下：

```
unsigned int a;                        // 定义一个变量
*P_IOA_GPIO_SETUP = 0x03030000;        // 设置 IOA0、IOA1 为输入口
a = *P_IOA_GPIO_INPUT;                 // 读取 IOA0、IOA1 的端口值
a &= 0x00000003;                       // 得到 IOA0、IOA1 的值
```

设置 IOA0、IOA1 为输出使能，输出高电平，参考程序段如下：

```
*P_IOA_GPIO_SETUP = 0x00000300;        // IOA0、IOA1 输出使能
*P_IOA_GPIO_SETUP |= 0x00000003;       // IOA0、IOA1 输出高电平
```

设置 IOA1 为上升沿外部中断，查询该中断发生，参考程序段如下：

```
*P_INT_MASK_CTRL2 &= ~C_INT_GPIO_DIS;               // 使能 IRQ 中断

*P_IOA_GPIO_INT = C_IOA1_INTRISE_EN                 // 使能 IOA1 上升沿外部中断
                | C_IOA1_INTRISE_FLAG;
while(1)
{
    if(*P_IOA_GPIO_INT & C_IOA1_INTRISE_FLAG)       // 判断中断是否发生
    {
        *P_IOA_GPIO_INT |= C_IOA1_INTRISE_FLAG;     // 清除中断标志位
    }
}
```

4.2 定时器——TIMER

4.2.1 概述

SPCE3200 内嵌 6 个 16 位 CCP（Compare、Capture、PWM，比较、捕获、PWM 输出）定时器：Timer0、Timer1、Timer2、Timer3、Timer4 和 Timer5。6 个定时器的功能结构完全相同，2 个时钟源可供选择 27MHz/(M+1)或 32768Hz 实时时钟。

4.2.2 特性

Timer0~Timer5 这 6 个定时器均可以编程设置工作方式及时钟源。可以编程设置为 normal 模式，即定时计数模式；CMP 模式，即比较模式；CAP 模式，即捕获模式；PWM 模式。每个定时器也可以单独设置时钟源，选择时钟源 27MHz/（M+1）或 32768Hz 实时时钟。

4.2.3 引脚描述

当定时器工作在比较输出模式、捕获输入模式和 PWM 输出模式时，均需要通过芯片管脚输入、输出数据，表 4.12 列出了这些管脚：

表 4.12 定时器 CCP 管脚描述

引脚名称	引脚号	引脚属性	引脚功能
USBDET	156	I/O	作为 Timer_CCP0 的输入输出
UART_RX	40	I/O	作为 Timer_CCP1 的输入输出
UART_TX	39	I/O	作为 Timer_CCP2 的输入输出
JTAG_TRSTN	153	I/O	作为 Timer_CCP3 的输入输出

引脚名称	引脚号	引脚属性	引脚功能
IOA0	41	I/O	作为 Timer_CCP4 的输入输出
IOA1	42	I/O	作为 Timer_CCP5 的输入输出

4.2.4 结构

定时器 Timer 的结构如图 4.3 所示：由于 6 个定时器的结构完全相同，以 Timer0 介绍定时器的结构。时钟源可以通过编程选择其一，16 位计数器在选择的频率下从计数初值计数，控制逻辑控制定时器 Timer 的使能、中断使能等，当工作在 CCP 模式下，CCP 寄存器单元作为计算 PWM 占空比、比较输出或捕获输入的存储单元。

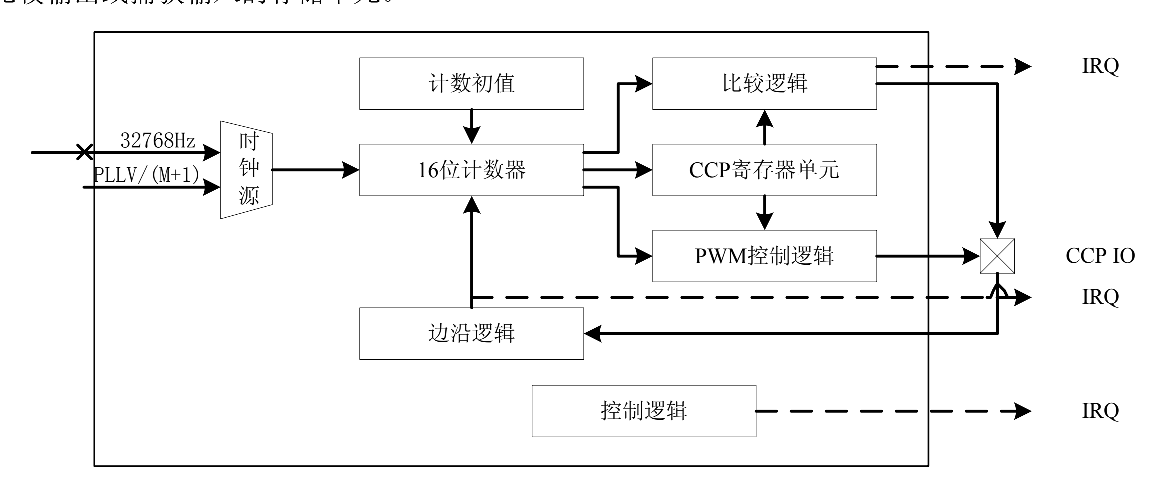


图 4.3 Timer 的结构

4.2.5 寄存器描述

由于 6 个定时器的结构、使用方法完全相同，这里以 Timer0 为例介绍。共有 9 个寄存器单元与定时器 Timer0 有关，参考表 4.13：

表 4.13 Timer0 相关的寄存器

寄存器中文名称	寄存器英文名称	地址
TIMER 时钟选择寄存器	P_TIMER_CLK_SEL	0x882100E4
TIMER 接口选择寄存器	P_TIMER_INTERFACE_SEL	0x88200010
TIMER0 时钟配置寄存器	P_TIMER0_CLK_CONF	0x8821006C
TIMER0 控制寄存器	P_TIMER0_MODE_CTRL	0x88160000
TIMER0 CCP 控制寄存器	P_TIMER0_CCP_CTRL	0x88160004
TIMER0 计数初值数据寄存器	P_TIMER0_PRELOAD_DATA	0x88160008
TIMER0 计数寄存器	P_TIMER0_COUNT_DATA	0x88160010
TIMER0 CCP 计数初值数据寄存器	P_TIMER0_CCP_DATA	0x8816000C

其中 P_TIMER_CLK_SEL、P_TIMER_INTERFACE_SEL 为 Timer0~Timer5 共用。下面分别介绍：

TIMER 时钟选择寄存器：P_TIMER_CLK_SEL(0x882100E4)

定时器时钟源选择寄存器，选择定时器使用 27M/（M+1）时钟源或者 32768Hz 时钟源，当选择 27M/（M+1）时钟源时，需要通过该寄存器的最低 8 位确定 M 值。当选择 32768Hz 时钟源时，需要使能 P_CLK_32K_CONF(0x88210114)寄存器。

表 4.14 P_TIMER_CLK_SEL(0x882100E4)

位	b13	b12	b11	b10	b9	b8	b7~b0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0	0
名称	Timer5C	Timer4C	Timer3C	Timer2C	Timer1C	Timer0C	DividedNum

Timer5C　b13　Timer5 时钟源选择位：
0：选择 27MHz 作为时钟源
1：选择 32768Hz 作为时钟源

Timer4C　b12　Timer4 时钟源选择位：
0：选择 27MHz 作为时钟源
1：选择 32768Hz 作为时钟源

Timer3C　b11　Timer3 时钟源选择位：
0：选择 27MHz 作为时钟源
1：选择 32768Hz 作为时钟源

Timer2C　b10　Timer2 时钟源选择位：
0：选择 27MHz 作为时钟源
1：选择 32768Hz 作为时钟源

Timer1C　b9　Timer1 时钟源选择位：
0：选择 27MHz 作为时钟源
1：选择 32768Hz 作为时钟源

Timer0C　b8　Timer0 时钟源选择位：
0：选择 27MHz 作为时钟源
1：选择 32768Hz 作为时钟源

DividedNum　b7~b0　时钟源采用 27MHz/(DividedNum + 1)，如果选择时钟源 27MHz

TIMER 接口选择寄存器：P_TIMER_INTERFACE_SEL(0x88200010)

当定时器工作在 CCP 模式下，需要设置该寄存器使 I/O 口作为 CCP 端口使用。

表 4.15 P_TIMER_INTERFACE_SEL(0x88200010)

位	b31~b22	b21	b20	b19	b18	b17	b16	b15~b0
读/写	-	R/W	R/W	R/W	R/W	R/W	R/W	-
默认值	-	0	0	0	0	0	0	-
名称	-	TCCP5	TCCP4	TCCP3	TCCP2	TCCP1	TCCP0	-

TCCP5　　b21　　IOA1 作为 CCP5 端口使能位：
0：不使能为 CCP5 端口
1：使能为 CCP5 端口

TCCP4　　b20　　IOA0 作为 CCP4 端口使能位：
0：不使能为 CCP4 端口
1：使能为 CCP4 端口

TCCP3　　b19　　JTAG_TRSTN 作为 CCP3 端口使能位：
0：不使能为 CCP3 端口
1：使能为 CCP3 端口

TCCP2　　b18　　UART_TX 作为 CCP2 端口使能位：
0：不使能为 CCP2 端口
1：使能为 CCP2 端口

TCCP1　　b17　　UART_RX 作为 CCP1 端口使能位：
0：不使能为 CCP1 端口
1：使能为 CCP1 端口

TCCP0　　b16　　USBDET 作为 CCP0 端口使能位：
0：不使能为 CCP0 端口
1：使能为 CCP0 端口

TIMER0 时钟配置寄存器 P_TIMER0_CLK_CONF(0x8821006C)

配置 Timer0 模块的时钟，当需要使用定时器 Timer0 模块时，需要配置此寄存器。

表 4.16 P_TIMER0_CLK_CONF(0x8821006C)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	TIMER0_RST	TIMER0_STOP

TIMER0_RST　b1　Timer0 模块时钟复位位：
0：Timer0 模块时钟复位
1：Timer0 模块时钟不复位

TIMER0_STOP　b0　Timer0 模块时钟停止位：
0：Timer0 模块时钟停止
1：Timer0 模块时钟使能

其它定时器的配置寄存器，参考表 4.17：各位功能与 P_TIMER0_CLK_CONF 相同。

表 4.17 定时器时钟配置寄存器

定时器 X	寄存器助记符	地址
定时器 0	P_TIMER0_CLK_CONF	0x8821006C
定时器 1	P_TIMER1_CLK_CONF	0x88210070
定时器 2	P_TIMER2_CLK_CONF	0x88210074
定时器 3	P_TIMER3_CLK_CONF	0x88210078
定时器 4	P_TIMER4_CLK_CONF	0x8821007C
定时器 5	P_TIMER5_CLK_CONF	0x88210080

TIMER0 控制寄存器：P_TIMER0_MODE_CTRL(0x88160000)

定时器模块 Timer0 的控制寄存器，使能 Timer0，使能 Timer0 中断及清除中断标志位。

表 4.18 P_TIMER0_MODE_CTRL(0x88160000)

位	b31	b30~b28	b27	b26	b25~b0
读/写	R/W	-	R/W	R/W	-
默认值	0	-	0	0	-
名称	TIMER0_EN	-	TIMER0_IRQ_EN	TIMER0_IRQ_FLAG	-

TIMER0_EN　b31　Timer0 使能位：
0：不使能 Timer0
1：使能 Timer0

TIMER0_IRQ_EN　b27　Timer0 中断使能位：
0：不使能 Timer0 中断
1：使能 Timer0 中断

TIMER0_IRQ_FLAG　b26　Timer0 中断标志位：
读 0：没有发生 Timer0 中断



读 1：发生 Timer0 中断
写 0：无意义
写 1：清除中断标志

其他定时器的控制寄存器参考表 4.21：各位功能与 P_TIMER0_MODE_CTRL 相同。

表 4.19 定时器控制寄存器

定时器 X	寄存器助记符	地址
定时器 0	P_TIMER0_MODE_CTRL	0x88160000
定时器 1	P_TIMER1_MODE_CTRL	0x88161000
定时器 2	P_TIMER2_MODE_CTRL	0x88162000
定时器 3	P_TIMER3_MODE_CTRL	0x88163000
定时器 4	P_TIMER4_MODE_CTRL	0x88164000
定时器 5	P_TIMER5_MODE_CTRL	0x88165000

TIMER0 CCP 控制寄存器：P_TIMER0_CCP_CTRL(0x88160004)

该寄存器可以设置定时器 0 的工作方式及在 CCP 工作模式下的选项。

表 4.20 P_TIMER0_CCP_CTRL(0x88160004)

位	b31~b30	b29~b28	b27	b26	b25	b24~b0
读/写	R/W	-	R/W	R/W	R/W	R/W
默认值	0	-	0	0	0	0
名称	CCP_MODE	-	CAP_MODE	COM_MODE	PWM_MODE	-

CCP_MODE　b31~b30　Timer0 工作方式选择位：
00：定时器模式
01：Capture 捕获输入模式
10：Compare 比较输出模式
11：PWM 输出模式

CAP_MODE　b27　CAP 输入模式选择位：
0：下降沿触发
1：上升沿触发

COM_MODE　b26　COM 输出模式选择位：
0：比较匹配后输出高脉冲
1：比较匹配后输出低脉冲

PWM_MODE　b25　　PWM 输出模式选择位：
0：NRO（Non-Return-One）输出模式
1：NRZ（Non-Return-Zero）输出模式

其他定时器的 CCP 控制寄存器如表 4.21：各位功能与 P_TIMER0_CCP_CTRL 相同。

表 4.21 定时器 CCP 控制寄存器

定时器 X	寄存器助记符	地址
定时器 0	P_TIMER0_CCP_CTRL	0x88160004
定时器 1	P_TIMER1_CCP_CTRL	0x88161004
定时器 2	P_TIMER2_CCP_CTRL	0x88162004
定时器 3	P_TIMER3_CCP_CTRL	0x88163004
定时器 4	P_TIMER4_CCP_CTRL	0x88164004
定时器 5	P_TIMER5_CCP_CTRL	0x88165004

TIMER0 计数初值数据寄存器：P_TIMER0_PRELOAD_DATA(0x88160008)

定时器的计数初值数据寄存器，16 位计数器将以该值计数，当溢出后会将该值重载入 16 位计数器。

表 4.22 P_TIMER0_PRELOAD_DATA(0x88160008)

位	b31~b16	b15~b0
读/写	-	R/W
默认值	-	0
名称	-	TIMER0_PRELOAD_REGS

其他定时器的计数初值寄存器如表 4.23：各位功能与 P_TIMER0_PRELOAD_DATA 相同。

表 4.23 定时器计数初值数据寄存器

定时器 X	寄存器助记符	地址
定时器 0	P_TIMER0_PRELOAD_DATA	0x88160008
定时器 1	P_TIMER1_PRELOAD_DATA	0x88161008
定时器 2	P_TIMER2_PRELOAD_DATA	0x88162008
定时器 3	P_TIMER3_PRELOAD_DATA	0x88163008
定时器 4	P_TIMER4_PRELOAD_DATA	0x88164008
定时器 5	P_TIMER5_PRELOAD_DATA	0x88165008



TIMER0 计数寄存器：P_TIMER0_COUNT_DATA(0x88160010)

定时器 0 的计数器，当定时器工作在定时/计数模式下时，可以读该寄存器得到当前计数值。

表 4.24 P_TIMER0_COUNT_DATA(0x88160010)

位	b31~b16	b15~b0
读/写	-	R/W
默认值	-	0
名称	-	TIMER0_UPCOUNT

其他定时器的计数器寄存器如表 4.25：各位功能与 P_TIMER0_COUNT_DATA 相同。

表 4.25 定时器计数寄存器

定时器 X	定时器助记符	地址
定时器 0	P_TIMER0_COUNT_DATA	0x88160010
定时器 1	P_TIMER1_COUNT_DATA	0x88161010
定时器 2	P_TIMER2_COUNT_DATA	0x88162010
定时器 3	P_TIMER3_COUNT_DATA	0x88163010
定时器 4	P_TIMER4_COUNT_DATA	0x88164010
定时器 5	P_TIMER5_COUNT_DATA	0x88165010

TIMER0 CCP 计数初值数据寄存器：P_TIMER0_CCP_DATA(0x8816000C)

当定时器 0 工作在 Compare 比较模式下，该寄存器的值与 16 位计数器的值作比较，当相等时输出高/低脉冲同时产生 IRQ 中断；当定时器 0 工作在 Capture 捕获输入模式下，当边沿逻辑捕获到周期信号后，将值保存到该寄存器，同时产生 IRQ 中断；当定时器 0 工作在 PWM 模式下，该寄存器与占空比有关。

表 4.26 P_TIMER0_CCP_DATA(0x8816000C)

位	b31~b16	b15~b0
读/写	-	R/W
默认值	-	0
名称	-	TIMER0_CCP_REGS

其他定时器的 CCP 计数初值数据寄存器参考：各位功能与 P_TIMER0_CCP_DATA 相同。

表 4.27 定时器计数初值数据寄存器

定时器 X	定时器助记符	地址

定时器 X	定时器助记符	地址
定时器 0	P_TIMER0_CCP_DATA	0x8816000C
定时器 1	P_TIMER1_CCP_DATA	0x8816100C
定时器 2	P_TIMER2_CCP_DATA	0x8816200C
定时器 3	P_TIMER3_CCP_DATA	0x8816300C
定时器 4	P_TIMER4_CCP_DATA	0x8816400C
定时器 5	P_TIMER5_CCP_DATA	0x8816500C

4.2.6 基本操作

定时/计数工作模式

Timer 用于定时/计数模式时的结构简化为如图 4.4 所示：

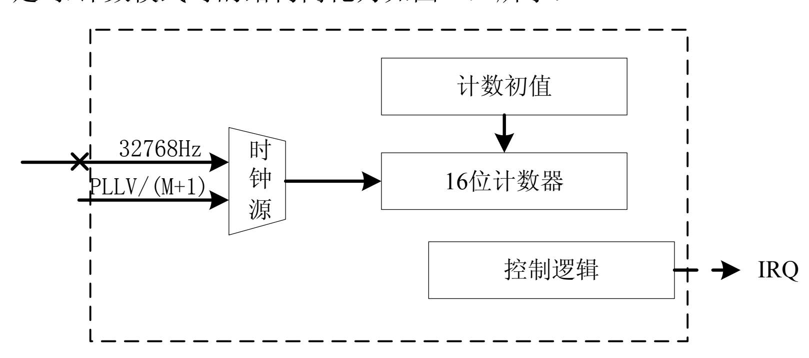


图 4.4 定时器 Timer 工作在定时/计数模式

Timer0~5 工作在定时/计数模式下的时序如图 4.5 所示：

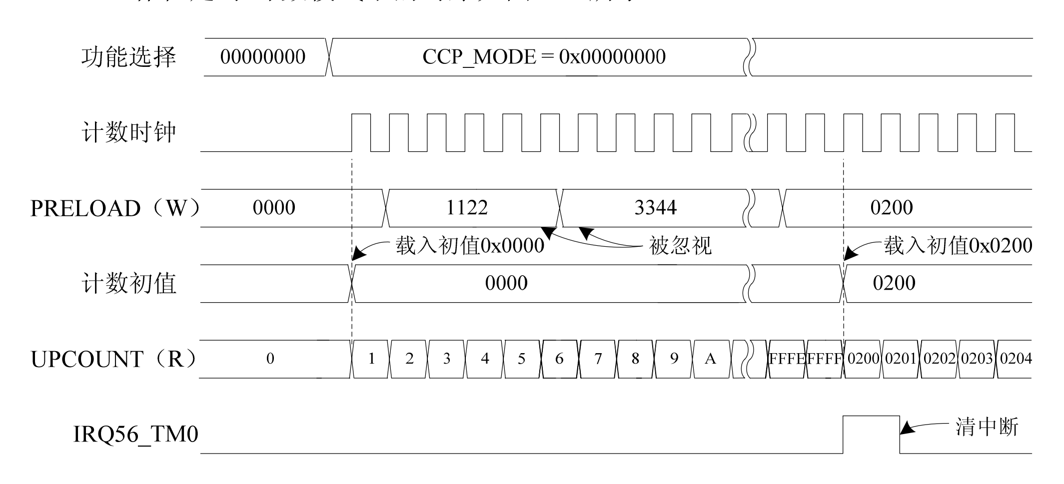


图 4.5 Timer0~5 定时/计数时序图

向 P_TIMER0_CCP_CTRL 寄存器的 b31:b30 中写入 00，定时器 0 将工作在定时/计数模式下。在使能定时器后，在下一个有效计数时钟沿（上升沿），计数初值将会被载入，之后在计数时钟作用下，计数器值递增，直至计数器溢出，将重载预置的计数初值，开始新一轮的计数。在定时器溢出之前，无法重新装载初值。

Timer 用于定时，只需选定适当的计数时钟源和计数初值即可。定时时间（T）与计数时钟频率



（f）、计数初值（N）的关系如下：

$$T = (65536 – N)/f$$

更多的情况是已知定时时间 T 和计数时钟频率 f，则计数初值 N 可以由下式算得：

$$N= 65536 – T * f$$

定时/计数器大多是与中断配合使用的。定时器的溢出信号可作为中断源，定时器作为定时/计数器的初始化流程如图 4.6 所示：

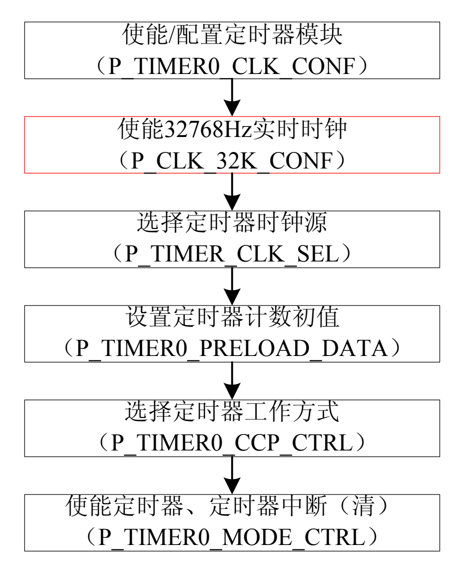


图 4.6 Timer0~5 定时/计数初始化流程

【例 4.1】：

利用 Timer0 实现 0.5s 定时，并控制 IOA1 外接的 LED，使该 LED 闪烁，硬件原理如图 4.7：

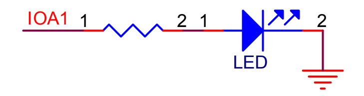


图 4.7 电路原理图

实现 0.5s 延时可以通过多种时钟源实现，这里采用 32768Hz，可计算得到计数初值为：

N = 65536 – 0.5 * 32768 = 49152(0xC000)

参考代码：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
    *P_TIMER0_CLK_CONF = C_TIMER_CLK_EN             // 使能 TIMER0 模块时钟
                        | C_TIMER_RST_DIS;
    *P_CLK_32K_CONF = C_32K_CRY_EN;                 // 使能 32K 晶振
```

```
        *P_TIMER_CLK_SEL = C_TIMER0_CLK_32K;            // 选择 32K 时钟
        *P_TIMER0_PRELOAD_DATA = 0xC000;                // 计数初值选择 0.5s
        *P_TIMER0_CCP_CTRL = C_TIMER_NOR_MODE;          // 工作模式选择定时计数模式
        *P_TIMER0_MODE_CTRL = C_TIMER_CTRL_EN           // 使能定时器
                            | C_TIMER_INT_EN            // 使能中断
                            | C_TIMER_INT_FLAG;         // 清中断
        *P_IOA_GPIO_SETUP = 0x00000300;                 // 使能 IOA 口位输出口

        while(1)
        {
            if(*P_TIMER0_MODE_CTRL & C_TIMER_INT_FLAG)
            {
                *P_IOA_GPIO_SETUP ^= 0x00000003;
            }
            *P_TIMER0_MODE_CTRL |= C_TIMER_INT_FLAG; // 清中断
        }
        return 0;
}
```

比较输出模式（Compare）

Timer0~5 用作比较输出的内部结构如图 4.8 所示：

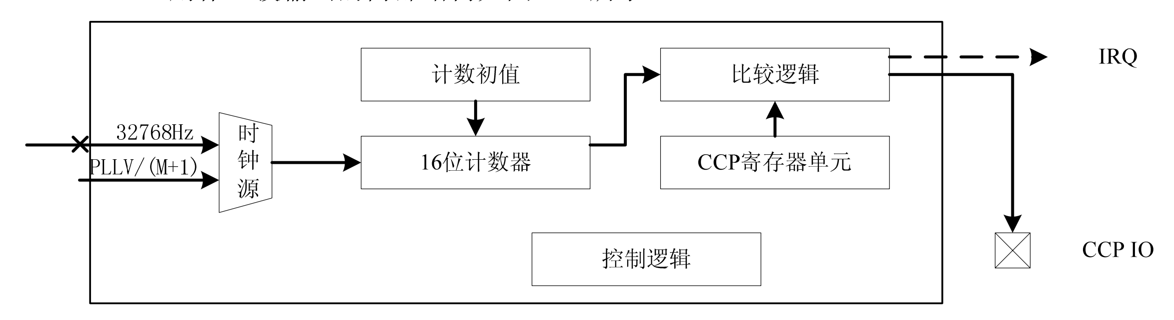


图 4.8 比较输出结构图

Timer0~5 在比较模式下的工作时序如图 4.9 所示：

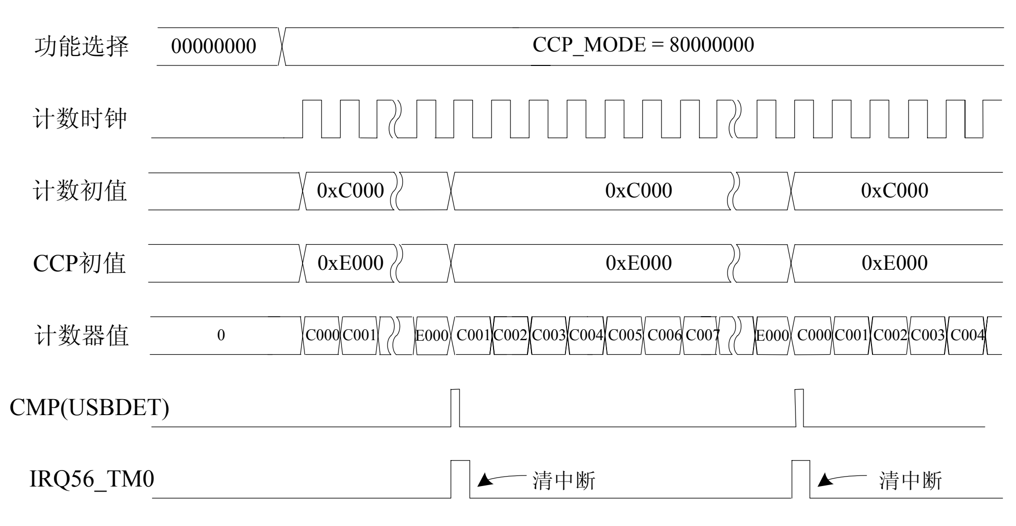


图 4.9 Timer0~5 比较输出时序图

Timer0~5 用作比较输出模式包括以下操作：配置定时器模块、选择定时器时钟源、设置定时器计数初值及 CCP 初值、使能 I/O 作为 CCP 端口使用、选择定时器工作方式为比较输出方式、使能定时器及中断。其操作流程如图 4.10：

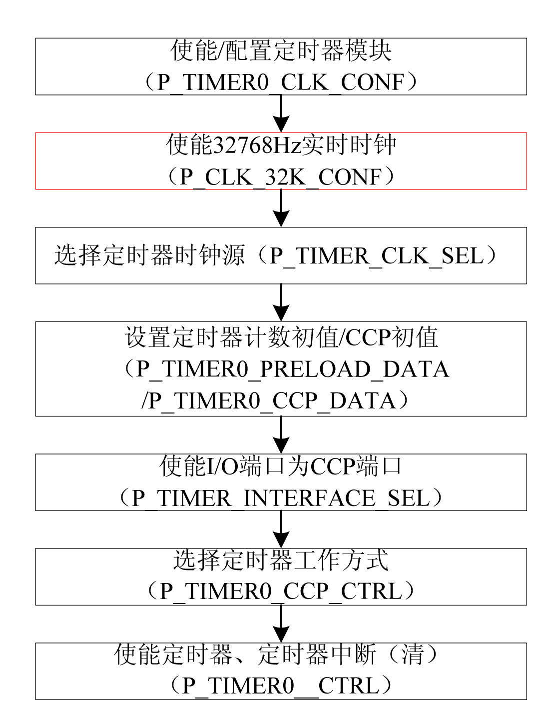


图 4.10 Timer0~5 比较输出初始化流程图

通过设置 P_TIMER0_CCP_CTRL 寄存器的 b26 位可以选择当比较匹配是输出高脉冲还是低脉冲。b26 为 0 输出高脉冲；b26 为 1 输出低脉冲。

【例 4.2】：设置比较输出，使用定时器 4 来完成。参考代码：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
```

```
int main(void)
{
        *P_INT_MASK_CTRL1 = ~C_INT_TIMER_DIS;                    // 使能 TIMER 中断
        *P_CLK_32K_CONF = C_32K_CRY_EN;                          // 使能 32K 晶振
        *P_TIMER4_CLK_CONF = 0x00000000;                         // 复位 TIMER4 模块
        *P_TIMER4_CLK_CONF = C_TIMER_CLK_EN | C_TIMER_RST_DIS;// 使能 TIMER4 模块
        *P_TIMER_CLK_SEL = C_TIMER4_CLK_32K;                     // 选择 32K 时钟源
        *P_TIMER4_PRELOAD_DATA = 0xF200;                         // 设置计数初值
        *P_TIMER4_CCP_DATA = 0xF700;                             // 设置 CCP 初值
        *P_TIMER_INTERFACE_SEL = C_TIMER4_PORT_SEL;              // 选择 CCP 端口
        *P_TIMER4_CCP_CTRL = C_TIMER_CMP_MODE
                                | C_TIMER_CMP_HIGH;            // 选择 CMP 工作模式
        *P_TIMER4_MODE_CTRL = C_TIMER_CTRL_EN                    // 使能 TIMER4
                                | C_TIMER_INT_EN                 // 使能中断
                                | C_TIMER_INT_FLAG;              // 清中断
        while(1);
        return 0;
}
// 中断文件中函数
//=============================================================
// 语法格式：void IRQ56(void)
// 功能描述: 定时器（Timer）中断服务函数
// 入口参数: 无
// 出口参数: 无
//=============================================================
void IRQ56(void)
{
        if(*P_TIMER4_MODE_CTRL & C_TIMER_INT_FLAG)
        {
                *P_TIMER4_MODE_CTRL |= C_TIMER_INT_FLAG;
        }
}
```

捕获输入模式（Capture）

Timer0~5 用作捕获输入模式（Capture）的结构如图 4.11 所示：当外部信号满足边沿逻辑的要求后，触发边沿逻辑，边沿逻辑控制 16 位计数器将捕获到的周期/频率信号输入 CCP 寄存器单元。

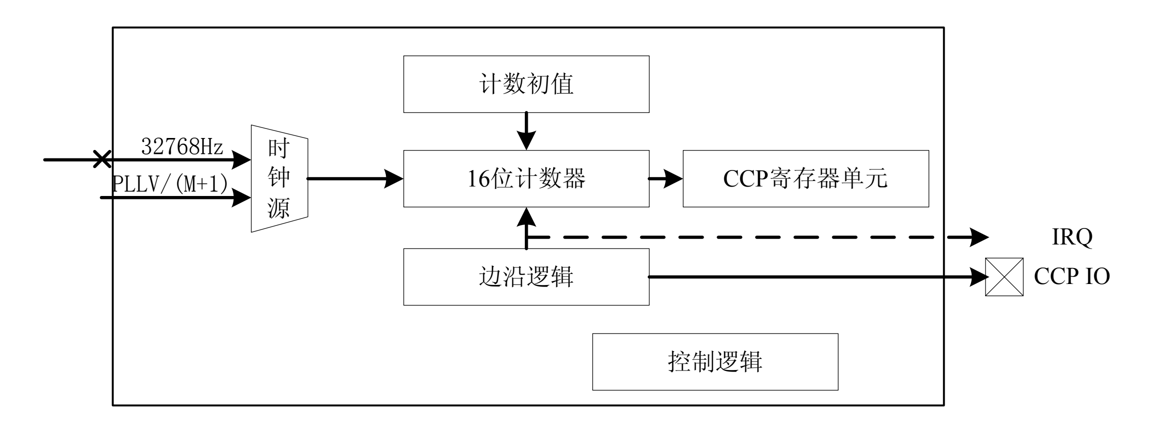


图 4.11 Timer0~5 捕获输入结构图

Capture 的时序图，这里设置上升沿触发，如图 4.12 所示：

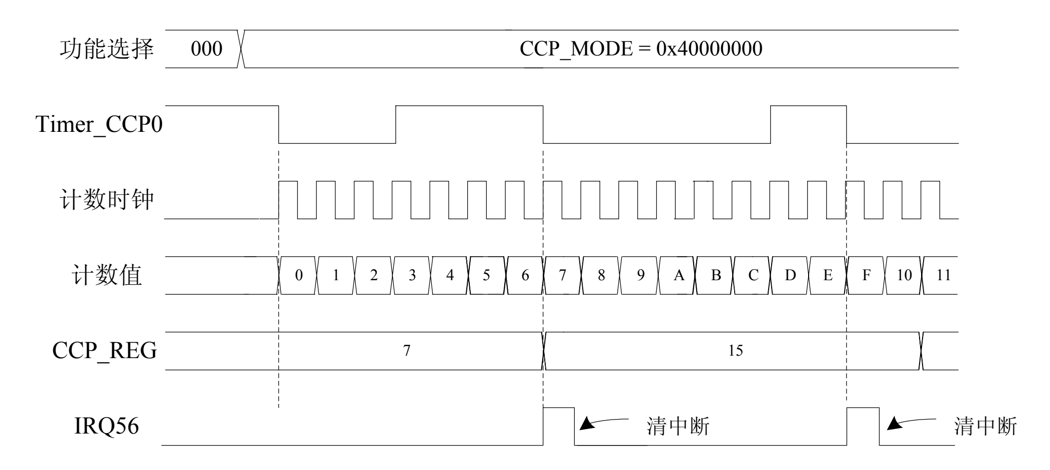


图 4.12 Timer0~5 捕获输入时序图

应用 Timer0~5 的捕获功能，需要以下操作：配置定时器模块、选择定时器时钟源、设置定时器计数初值及 CCP 初值、使能 I/O 作为 CCP 端口使用、选择定时器工作方式为比较捕获方式、使能定时器及中断。其操作流程如图 4.13：

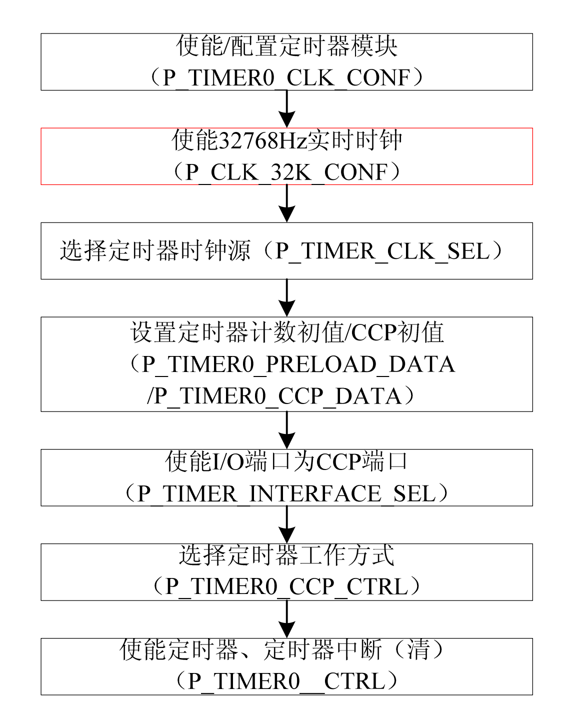


图 4.13 Timer0~5 捕获输入初始化流程图

【例 4.3】：使用定时器 5 作为 PWM 输出，定时器 4 作为比较捕获输入，获得 PWM 的输出周期。参考代码：

```
#include "SPCE3200_Constant.h"
#include "SPCE3200_Register.h"
//-----------------------------------------主函数---------------------------------------//
int main(void)
{
    *P_INT_MASK_CTRL1 = ~C_INT_TIMER_DIS;           // 打开 TIMER 中断
    *P_CLK_32K_CONF = C_32K_CRY_EN;                 // 打开 32K 晶振
    *P_TIMER_CLK_SEL = C_TIMER4_CLK_32K
                   | C_TIMER5_CLK_32K;              // 选择 TIMER4、TIMER5 时钟源
    *P_TIMER_INTERFACE_SEL = C_TIMER4_PORT_SEL
                   | C_TIMER5_PORT_SEL;

    *P_TIMER5_CLK_CONF = 0;                         // 复位 TIMER5 模块
    *P_TIMER5_CLK_CONF = C_TIMER_CLK_EN
                   | C_TIMER_RST_DIS;               // 使能 TIMER5 模块
    *P_TIMER5_PRELOAD_DATA = 0xF200;                // 设置计数初值
    *P_TIMER5_CCP_DATA = 0xF700;                    // 设置 CCP 初值
    *P_TIMER5_CCP_CTRL = C_TIMER_PWM_MODE;          // 选择 TIMER5 工作在 PWM 模式
```

```
        *P_TIMER5_MODE_CTRL = C_TIMER_CTRL_EN         // 使能 TIMER5
                          | C_TIMER_INT_EN | C_TIMER_INT_FLAG;

        *P_TIMER4_CLK_CONF = 0;                       // 复位 TIMER4 模块
        *P_TIMER4_CLK_CONF = C_TIMER_CLK_EN
                          | C_TIMER_RST_DIS;          // 使能 TIMER4 模块
        *P_TIMER4_CCP_CTRL = C_TIMER_CAP_MODE         // 选择 TIMER4 工作在捕获模式
                          | C_TIMER_CAP_FALL;
        *P_TIMER4_MODE_CTRL = C_TIMER_CTRL_EN         // 使能 TIMER4
                          | C_TIMER_INT_EN | C_TIMER_INT_FLAG;

        while(1);
        return 0;
}
//------------------------------------------中断服务函数----------------------------------------//
IRQ56(void)
{
        unsigned int t;
        unsigned int h;
        if(*P_TIMER5_MODE_CTRL & C_TIMER_INT_FLAG)
        {
                *P_TIMER5_MODE_CTRL |= C_TIMER_INT_FLAG;
        }
        if(*P_TIMER4_MODE_CTRL & C_TIMER_INT_FLAG)
        {
                *P_TIMER4_MODE_CTRL |= C_TIMER_INT_FLAG;
                t = *P_TIMER4_COUNT_DATA;
                h = *P_TIMER4_CCP_DATA;               // 获得 PWM 周期

        }
}
```

PWM 输出模式

Timer0~5 作 PWM 输出的结构如图 4.14 所示：其核心是一个 16 位计数器与 PWM 控制逻辑。通过 P_TIMER0_PRELOAD_DATA 与 P_TIMER0_CCP_DATA 寄存器决定 TIMER0 模块的 PWM 信号的周期和脉宽，PWM 信号通过 CCP IO 输出。具体对应芯片上管脚参考表 4.4。

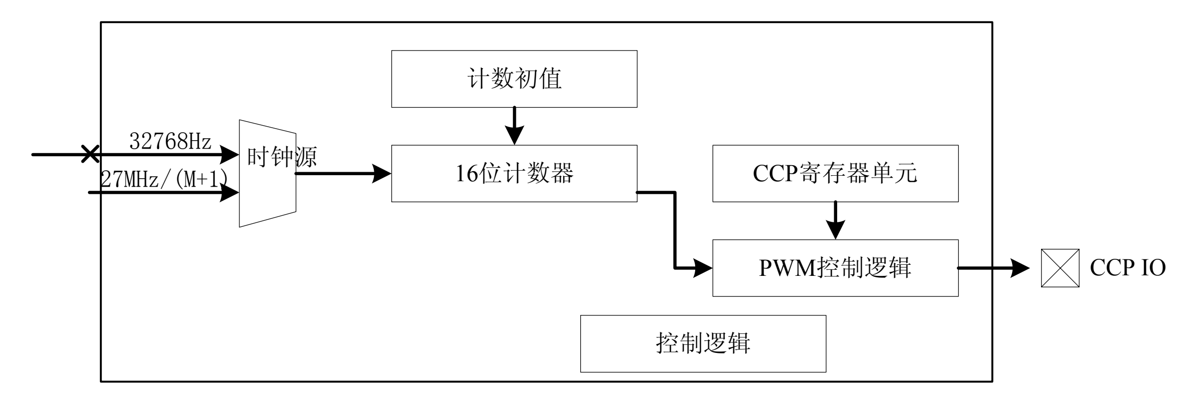


图 4.14 Timer0~5 PWM 输出结构图

Timer0~5 PWM 输出的时序如图 4.15 所示：

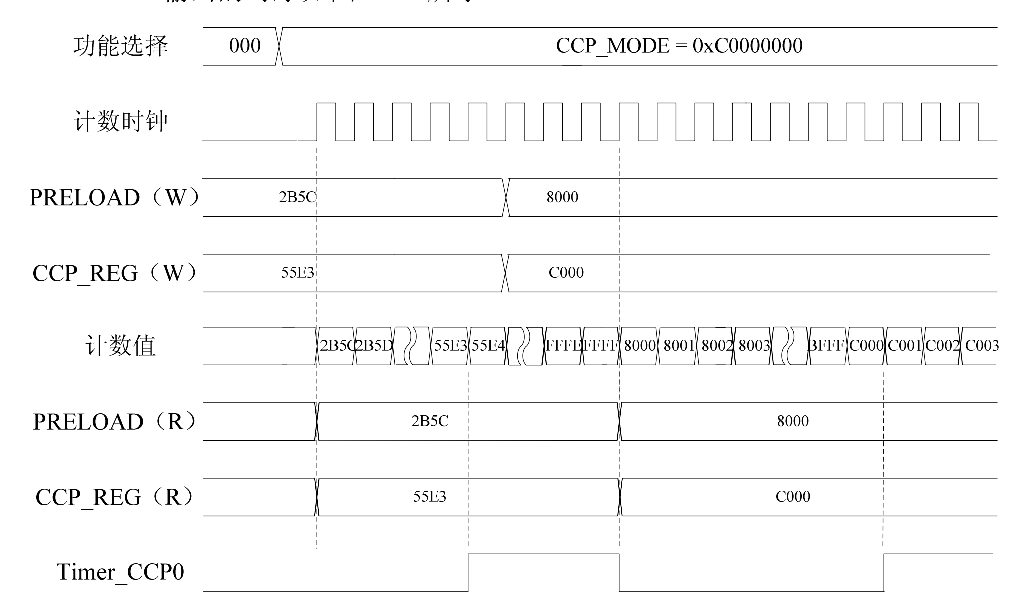


图 4.15 Timer0~5 PWM 输入时序图

向 P_TIMER0_CCP_CTRL 寄存器的 b31:b30 写入 11，选择定时器工作在 PWM 输出模式下，使能定时器，计数器载入 P_TIMER0_PRELOAD_DATA 与 P_TIMER0_CCP_DATA 的值，并开始计数。当计数值 P_TIMER0_COUNT_DATA 与 P_TIMER0_CCP_DATA 的值匹配时置位 PWM 输出端口，当计数器溢出时（达到 65536）清除 PWM 输出端口，周而复始。当计数器溢出时，可以重新载入计数初值与 CCP 初值。若定时器的计数时钟频率为 f，P_TIMER0_PRELOAD_DATA 和 P_TIMER0_CCP_DATA 预置的值分别为 PRELOAD 和 CCP_REG，则输出 PWM 信号的周期和占空比由图 4.16 公式计算：

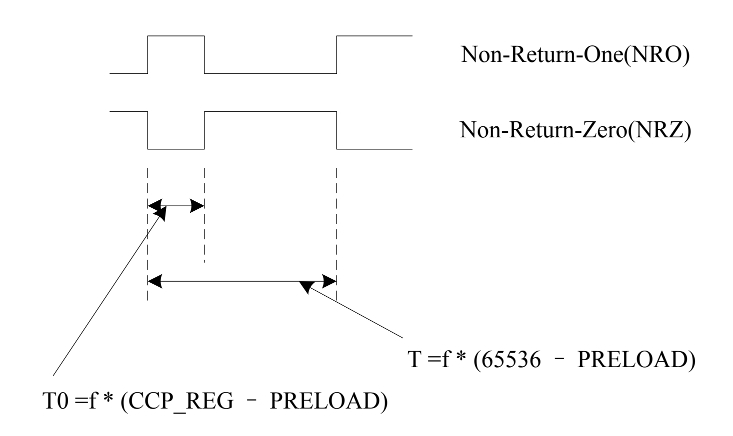


图 4.16 PWM 输出信号计算方法

通过设置寄存器 P_TIMER0_CCP_CTRL 的 b25 位选择 NRO 模式或 NRZ 模式，当 b25 为 0 时选择 NRO 模式；当 b25 为 1 时选择 NRZ 模式。

应用 Timer0~5 的 PWM 输出功能，需要以下操作：配置定时器模块、选择定时器时钟源、设置定时器计数初值及 CCP 初值、使能 I/O 作为 CCP 端口使用、选择定时器工作方式为 PWM 输出方式、使能定时器及中断。其操作流程如下：

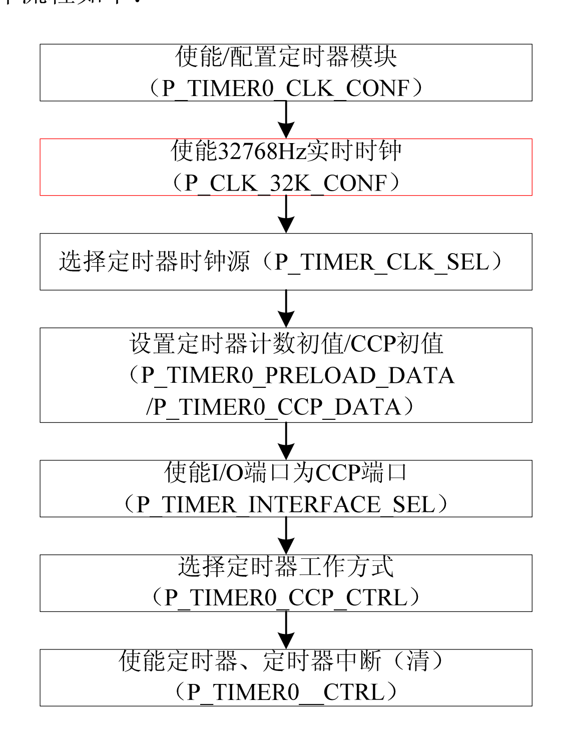
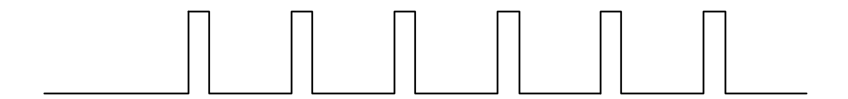


图 4.17 Timer0~5　PWM 输出初始化流程图

【例 4.4】：

利用 Timer0 的 PWM 输出功能，通过 Timer0_CCP 输出周期为 0.5s，占空比为 20%的波形（FSYS = 32768Hz）。



根据公式 T = f * (65536 - PRELOAD)和 T0 = f * (CCP_REG - PRELOAD)计算得到 PRELOAD =

0xC000，CCP_REG = 0xCCCC。

参考代码：

```
#include "SPCE3200_Constant.h"
#include "SPCE3200_Register.h"

void PWM_OUT(unsigned int preload, unsigned int valueccp);
void PWM_OUT(unsigned int preload, unsigned int valueccp)
{
    *P_INT_MASK_CTRL1 = ~C_INT_TIMER_DIS;                  // 打开 TIMER 中断
    *P_TIMER0_CLK_CONF = 0;                                // TIMER0 模块复位
    *P_TIMER0_CLK_CONF = C_TIMER_CLK_EN                    // TIMER0 模块使能
                    | C_TIMER_RST_DIS;
    *P_CLK_32K_CONF = C_32K_CRY_EN;                        // 使能 32K 晶振
    *P_TIMER_CLK_SEL = C_TIMER0_CLK_32K;                   // TIMER0 选择 32k 时钟源
    *P_TIMER_INTERFACE_SEL = C_TIMER0_PORT_SEL;            // 使能 TIMER0 CCP 端口
    *P_TIMER0_PRELOAD_DATA = preload;                      // 载入初值
    *P_TIMER0_CCP_DATA = valueccp;                         // 载入 CCP 初值
    *P_TIMER0_CCP_CTRL = C_TIMER_PWM_MODE;                 // TIMER0 选择 PWM 工作模式
    *P_TIMER0_MODE_CTRL = C_TIMER_CTRL_EN                  // TIMER0 使能、中断使能
                    | C_TIMER_INT_EN
                    | C_TIMER_INT_FLAG;
}

int main(void)
{
    PWM_OUT(0xC000,0xCCCC);
    while(1);
    return 0;
}
```

4.2.7 注意事项

TIMER 在使用时需要将该模块先复位在使能，然后才能使用。



4.3 实时时钟——RTC

4.3.1 概述

实时时钟 RTC（Real Time Clock）提供一套计时系统，由 32768Hz 的时钟源经过分频后得到秒、分、时以及闹钟信号，可以方便的得到计时时间。

4.3.2 特征

- 提供 1/2 秒、1 秒、1 分以及 1 小时的中断请求
- 提供闹钟信号中断请求

4.3.3 寄存器描述

和 RTC 有关的寄存器总共有 10 个，参考表 4.28：

表 4.28 RTC 相关寄存器描述

寄存器英文名称	寄存器中文名称	地址
32K 实时时钟配置寄存器	P_CLK_32K_CONF	0x88210114
RTC 时钟配置寄存器	P_RTC_CLK_CONF	0x88210088
RTC 秒寄存器	P_RTC_TIME_SEC	0x88166000
RTC 分寄存器	P_RTC_TIME_MIN	0x88166004
RTC 时寄存器	P_RTC_TIME_HOUR	0x88166008
RTC 秒报警寄存器	P_RTC_ALM_SEC	0x8816600C
RTC 分报警寄存器	P_RTC_ALM_MIN	0x88166010
RTC 时报警寄存器	P_RTC_ALM_HOUR	0x88166014
RTC 控制寄存器	P_RTC_MODE_CTRL	0x88166018
RTC 中断状态寄存器	P_RTC_INT_STATUS	0x8816601C

32K 实时时钟配置寄存器：P_CLK_32K_CONF(0x88210114)

请参考第三章设置。

RTC 时钟配置寄存器：P_RTC_CLK_CONF(0x88210088)

配置 RTC 模块时钟的寄存器，当需要使用 RTC 模块，需要配置此寄存器。

表 4.29 P_RTC_CLK_CONF(0x88210088)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0

名称	-	RTC_RST	RTC_STOP

RTC_RST　b1　　RTC 模块时钟复位位：
0：RTC 模块时钟复位
1：RTC 模块时钟不复位

RTC_STOP　b0　　RTC 模块时钟使能位：
0：RTC 模块时钟停止
1：RTC 模块时钟使能

RTC 秒寄存器：P_RTC_TIME_SEC(0x88166000)

初始化 RTC 计时系统的秒计时，该值范围在 0~59，如果写入其他值，将得到不可预知的结果。写入秒初始之后，当启动 RTC，该值每隔 1s 加一，直到 59，然后会使 P_RTC_TIME_MIN 寄存器的值加一，同时自身恢复为 00，周而复始。读该寄存器会得到当前的秒计时。

表 4.30 P_RTC_TIME_SEC(0x88166000)

位	**b5~b0**
读/写	R/W
默认值	0
名称	SECOND_SET

RTC 分寄存器：P_RTC_TIME_MIN(0x88166004)

初始化 RTC 计时系统的分计时，该值范围在 0~59，如果写入其他值，将得到不可预知的结果。写入分初始之后，当启动 RTC，该值每隔 60s 加一，直到 59，然后会使 P_RTC_TIME_HOUR 寄存器的值加一，同时自身恢复为 00，周而复始。读该寄存器会得到当前的分计时。

表 4.31 P_RTC_TIME_MIN(0x88166004)

位	**b5~b0**
读/写	R/W
默认值	0
名称	MINUTE_SET

RTC 时寄存器：P_RTC_TIME_HOUR(0x88166008)

初始化 RTC 计时系统的时计时，该值范围在 0~59，如果写入其他值，将得到不可预知的结果。写入时初始之后，当启动 RTC，该值每隔 3600s 加一，直到 59，自身恢复为 00，周而复始。读该寄存器会得到当前的小时。

表 4.32 P_RTC_TIME_HOUR(0x88166008)

位	**b5~b0**

读/写	R/W
默认值	0
名称	HOUR_SET

RTC 秒报警寄存器：P_RTC_ALM_SEC(0x8816600C)

初始化 RTC 定时系统的秒初值，该值范围在 0~59，如果写入其他值，将得到不可预知的结果。当计时系统的小时、分钟、秒钟与定时系统的小时、分钟、秒钟完全匹配时，产生定时报警中断。

表 4.33 P_RTC_ALM_SEC(0x8816600C)

位	**b5~b0**
读/写	R/W
默认值	0
名称	ALM_SECOND_SET

RTC 分报警寄存器：P_RTC_ALM_MIN(0x88166010)

初始化 RTC 定时系统的分初值，该值范围在 0~59，如果写入其他值，将得到不可预知的结果。当计时系统的小时、分钟、秒钟与定时系统的小时、分钟、秒钟完全匹配时，产生定时报警中断。

表 4.34 P_RTC_ALM_MIN(0x88166010)

位	**b5~b0**
读/写	R/W
默认值	0
名称	ALM_MINUTE_SET

RTC 时报警寄存器：P_RTC_ALM_HOUR(0x88166014)

初始化 RTC 定时系统的时初值，该值范围在 0~59，如果写入其他值，将得到不可预知的结果。当计时系统的小时、分钟、秒钟与定时系统的小时、分钟、秒钟完全匹配时，产生定时报警中断。

表 4.35 P_RTC_ALM_HOUR(0x88166014)

位	**b5~b0**
读/写	R/W
默认值	0
名称	ALM_HOUR_SET

RTC 控制寄存器：P_RTC_MODE_CTRL(0x88166018)

当使用 RTC 实时时钟模块时，需要设置此寄存器使能。

表 4.36 P_RTC_ALM_HOUR(0x88166014)

位	b31
读/写	R/W
默认值	0
名称	RTC_EN

RTC_EN　　b0　　　RTC 实时时钟使能位：

0: 不使能 RTC 实时时钟

1: 使能 RTC 实时时钟

RTC 中断状态寄存器：P_RTC_INT_STATUS(0x8816601C)

该寄存器可以使能 RTC 中断、判断 RTC 中断是否发生及清除 RTC 中断。

表 4.37 P_RTC_INT_STATUS(0x8816601C)

位	b31	b30	b29~b28	b27	b26	b25~b24	b23
读/写	R/W	R/W	-	R/W	R/W	-	R/W
默认值	0	0	-	0	0	-	0
名称	ALM_INT	ALM_INTEN	-	HOUR_INT	HOUR_INTEN	-	MIN_INT
位	b22	b21~b20	b19	b18	b17~b16	b15	b14
读/写	R/W	-	R/W	R/W	-	R/W	R/W
默认值	0	-	0	0	-	0	0
名称	MIN_INTEN	-	SEC_INT	SEC_INTEN	-	HSEC_INT	HSEC_INTEN

ALM_INT　　b31　　　闹钟中断标志位：

读 0：没有发生闹钟中断

读 1：发生闹钟中断

写 0：无意义

写 1：清除中断标志

ALM_INTEN　　b30　　　闹钟中断使能位：

0：不使能闹钟中断

1：使能闹钟中断

HOUR_INT　　b27　　　时中断标志位：

读 0：没有发生时中断

读 1：发生时中断

写 0：无意义

写 1：清除中断标志

HOUR_INTEN b26 时中断使能位：

0：不使能时中断

1：使能时中断

MIN_INT b23 分中断标志位：

读 0：没有发生分中断

读 1：发生分中断

写 0：无意义

写 1：清除中断标志

MIN_INTEN b22 分中断使能位：

0：不使能分中断

1：使能分中断

SEC_INT b19 秒中断标志位：

读 0：没有发生秒中断

读 1：发生秒中断

写 0：无意义

写 1：清除中断标志

SEC_INTEN b18 秒中断使能位：

0：不使能秒中断

1：使能秒中断

HSEC_INT b15 半秒中断标志位：

读 0：没有发生半秒中断

读 1：发生半秒中断

写 0：无意义

写 1：清除中断标志

HSEC_INTEN b14 半秒中断使能位：

0：不使能半秒中断

1：使能半秒中断

4.3.4 基本操作

当需要使用 RTC 模块时，需要按照下面流程初始化，如图 4.18 所示：首先需要配置 RTC 模块，然后使能 32768Hz 实时时钟，当需要采用计时系统时，初始化时间；当需要定时时，初始化定时；当只需要半秒、秒等中断时，不用初始化时间，默认为 0 时 0 分 0 秒，然后使能中断及使能 RTC 模块，初始化流程结束。

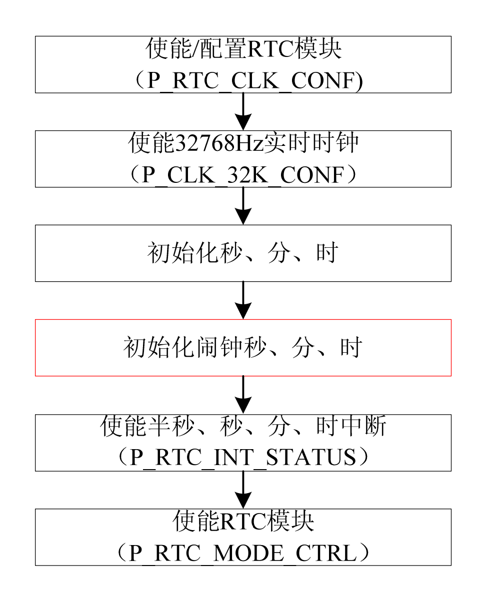
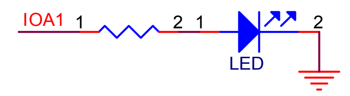


图 4.18 初始化 RTC 流程

4.3.5 应用举例

【例 4.5】：使用 RTC 半秒中断闪烁 LED。

硬件连接图如下：



参考代码：该代码使用查询中断方式实现。

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
    *P_IOA_GPIO_SETUP = 0x00000300;
    *P_CLK_32K_CONF = C_32K_CRY_EN;                          // 使能 32768Hz 晶振
    *P_RTC_CLK_CONF = C_RTC_CLK_EN | C_RTC_RST_DIS;          // RTC 时钟配置
    *P_RTC_MODE_CTRL = C_RTC_CTRL_EN;                        // 使能 RTC 模块
    *P_RTC_INT_STATUS = C_RTC_HSEC_INTEN                     // 使能半秒中断
                        | C_RTC_HSEC_FLAG;
    while(1)
    {
        if(*P_RTC_INT_STATUS & C_RTC_HSEC_FLAG)
        {
            *P_IOA_GPIO_SETUP ^= 0x00000003;
```

```
                    *P_RTC_INT_STATUS |= C_RTC_HSEC_FLAG;          //清中断标志
                }
            }
        }
```

4.4 时基——Time Base

4.4.1 概述

SPCE3200 内嵌时间基准信号发生器，简称时基。在 SPCE3200 内部共有这样的模块 3 个，TMB0、TMB1、TMB2，它们的结构完全相同，下文将以 TMB0 进行介绍。TMB0 模块通过一个计数器对 32768Hz 时钟信号进行分频，一共产生 12 个可操作时基信号：2048Hz、1024Hz、512Hz、256Hz、128Hz、64Hz、32Hz、16Hz、8Hz、4Hz、2Hz、1Hz 信号。

4.4.2 结构

TMB0 时基模块的结构如图 4.19 所示：设置 P_TMB_MODE_CTRL 寄存器可以选择时基的频率，在频率信号到来时产生 IRQ 中断。向 P_TMB_RESET_COMMAND 寄存器写入 0x5xxxxxx5 可以复位时基模块，在系统复位时也可以复位时基模块。

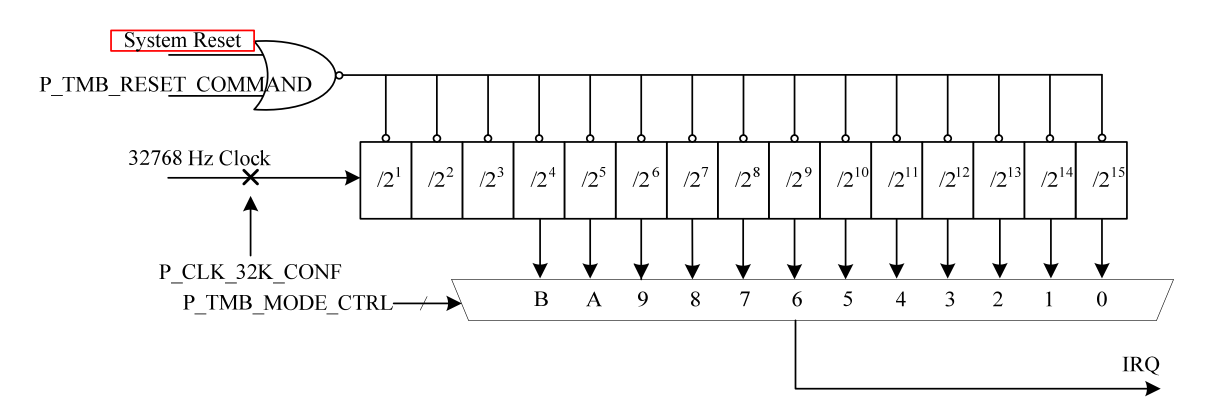


图 4.19 时基模块的结构

4.4.3 寄存器描述

与 TMB 相关的寄存器共有 5 个，参考表 4.38：

表 4.38 与 TMB 相关的寄存器

寄存器名称	助记符	地址
32K 实时时钟配置寄存器	P_CLK_32K_CONF	0x88210114
TMB 时钟配置寄存器	P_TMB_CLK_CONF	0x882100E0
TMB 控制寄存器	P_TMB_MODE_CTRL	0x88166020
TMB 中断状态寄存器	P_TMB_INT_STATUS	0x88166024
TMB 复位命令寄存器	P_TMB_RESET_COMMAND	0x88166028

TMB 时钟配置寄存器：P_TMB_CLK_CONF(0x882100E0)

配置 TMB 模块时钟寄存器，当需要使用 TMB 模块，需要配置此寄存器。

表 4.39 P_TMB_CLK_CONF(0x882100E0)

位	b31~b3	b2	b1	b0
读/写	-	R/W	-	R/W
默认值	-	0	-	0
名称	-	TBASE_RST	-	TBASE_STOP

TBASE_RST b2 TMB 模块时钟复位位：

0：TMB 模块时钟复位

1：TMB 模块时钟不复位

TBASE_STOP b0 TMB 模块时钟使能位：

0：TMB 模块时钟停止

1：TMB 模块时钟使能

TMB 控制寄存器：P_TMB_MODE_CTRL(0x88166020)

通过该寄存器可以使能 TMB0、TMB1 及 TMB2 模块，可以选择 TMB0、TMB1、TMB2 的时基频率。

表 4.40 P_TMB_MODE_CTRL(0x88166020)

位	b31	b27~b24	b23	b19~b16	b15	b11~b8
读/写	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0
名称	TMB0_EN	TMB0_SEL	TMB1_EN	TMB1_SEL	TMB2_EN	TMB2_SEL

TMB0_EN b31 TMB0 使能位：

0：TMB0 使能

1：TMB0 不使能

TMB0_SEL b27~b24 TMB0 输出时基频率选择位：

0000：1Hz

0001：2Hz

0010：4Hz

0011：8Hz

0100：16Hz

0101：32Hz

0110：64Hz

0111：128Hz

1000：256Hz

1001：512Hz

1010：1024Hz

1011：2048Hz

TMB1_EN b23 TMB1 使能位：

0：TMB1 使能

1：TMB1 不使能

TMB1_SEL b19~b16 TMB1 输出时基频率选择位：

0000：1Hz

0001：2Hz

0010：4Hz

0011：8Hz

0100：16Hz

0101：32Hz

0110：64Hz

0111：128Hz

1000：256Hz

1001：512Hz

1010：1024Hz

1011：2048Hz

TMB2_EN b15 TMB2 使能位：

0：TMB2 使能

1：TMB2 不使能

TMB2_SEL b11~b8 TMB2 输出时基频率选择位：

0000：1Hz

0001：2Hz

0010：4Hz

0011：8Hz

0100：16Hz

0101：32Hz

0110：64Hz

0111：128Hz

1000：256Hz

1001：512Hz

1010：1024Hz

1011：2048Hz

TMB 中断状态寄存器：P_TMB_INT_STATUS(0x88166024)

该寄存器使能 TMB0、TMB1、TMB2 的中断、判断中断是否发生及清除中断标志位。

表 4.41 P_TMB_INT_STATUS(0x88166024)

位	b31	b30	b27	b26	b23	b22
读/写	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0
名称	TMB0_INT	TMB0_INTEN	TMB1_INT	TMB1_INTEN	TMB2_INT	TMB2_INTEN

TMB0_INT　b31　TMB0 中断标志位：

读 0：没有发生 TMB0 中断

读 1：发生 TMB0 中断

写 0：无意义

写 1：清除中断标志

TMB0_INTEN　b30　TMB0 中断使能位：

0：不使能 TMB0 中断

1：使能 TMB0 中断

TMB1_INT　b27　TMB1 中断标志位：

读 0：没有发生 TMB1 中断

读 1：发生 TMB1 中断

写 0：无意义

写 1：清除中断标志

TMB1_INTEN　b26　TMB1 中断使能位：

0：不使能 TMB1 中断

1：使能 TMB1 中断

TMB2_INT　b23　TMB2 中断标志位：

读 0：没有发生 TMB2 中断

读 1：发生 TMB2 中断

写 0：无意义

写 1：清除中断标志

TMB2_INTEN　b22　TMB2 中断使能位：



0：不使能 TMB2 中断

1：使能 TMB2 中断

TMB 复位命令寄存器：P_TMB_RESET_COMMAND(0x88166028)

向该寄存器写入 0x5xxxxxx5 可以复位 TMB 时基计数器。

表 4.42 P_TMB_RESET_COMMAND(0x88166028)

位	**b31~b0**
读/写	R/W
默认值	0
名称	RESET_COM

RESET_COM　b31~b0　　写 0x5xxxxxx5 复位时基计数器

4.4.4　基本操作

当选择 TMB 模块工作时，初始化流程如图 4.20：首先配置 TMB 模块，然后使能 32768Hz 时钟，使能 TMB 的中断，选择 TMB 的时基频率，使能 TMB 模块。



图 4.20 初始化 TMB 流程

4.4.5　应用举例

【例 4.6】：使用 TM0 时基模块 2Hz 中断闪烁 LED。

硬件连接图如图 4.21：

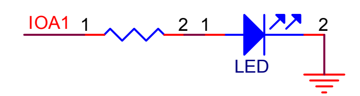


图 4.21 硬件连接图

参考代码：该代码使用查询中断方式实现。

```
#include "RTC.h"
```

```
int main(void)
{
    *P_CLK_32K_CONF = C_32K_CRY_EN;                          // 32768Hz 晶振使能
    *P_IOA_GPIO_SETUP = 0x00000202;
    *P_TMB_CLK_CONF = 0x00000000;                            // 复位时基模块
    *P_TMB_CLK_CONF = C_TMB_CLK_EN | C_TMB_RST_DIS;          // 时基时钟配置
    *P_TMB_INT_STATUS = C_TMB_TMB0_INTEN                     // 使能 TMB0 中断
                      | C_TMB_TMB0_FLAG;                     //
    *P_TMB_MODE_CTRL = C_TMB_TMB0_EN                         // 使能 TMB0
                      | C_TMB_TMB0_2HZ;                      // 选择 2Hz

    while(1)
    {
        while(*P_TMB_INT_STATUS & C_TMB_TMB0_FLAG)
        {
            * P_TMB_INT_STATUS |= C_TMB_TMB0_FLAG;           // 清中断标志
            * P_IOA_GPIO_SETUP ^= 0x00000002;
        }
    }
}
```

4.5 看门狗——WDOG

4.5.1 概述

为了避免因程序"跑飞"而引起的系统问题，SPCE3200 内部集成了看门狗计数器 WDOG（Watchdog）。

SPCE3200 的 WDOG 事实上就是一个 32 位的计数器，时钟由 32768Hz 的晶振提供。计数器递减计数，当计数器计到 0 时计数器溢出，引发系统复位。因此在使能看门狗后，在计数器还没有溢出前要及时喂狗，否则系统会因看门狗产生复位。

4.5.2 特性

WDOG 的特性如下：

- WDOG 可编程使能：即 WDOG 是否被使能可通过软件设置
- WDOG 的计数器递减计数
- WDOG 的计数初值可编程设置：即 WDOG 计数器的计数初值可通过软件来设置
- WDOG 可通过喂狗来重载计数初值：使能看门狗后，如果执行喂狗操作，WDOG 计数器重载计数初值

4.5.3 结构

WDOG 的结构框图如图 4.22：32768Hz 晶振直接给看门狗计数器提供时钟；可以通过寄存器设置看门狗计数器的计数初值；使能看门狗后，看门狗计数器载入计数初值开始递减计数；在计数的过程中，如果执行了喂狗操作，看门狗计数器重载计数初值开始递减计数；一直不喂狗会引发异常并使系统复位。

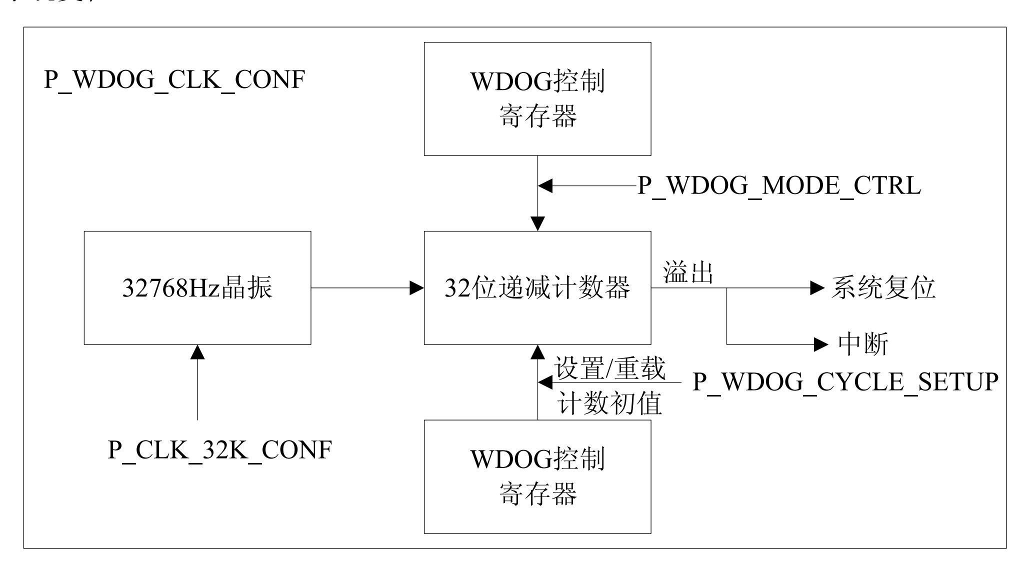


图 4.22 WDOG 的结构框图

4.5.4 寄存器描述

看门狗模块相关的寄存器共有 6 个，如表 4.43。通过这些寄存器可以打开或者关闭看门狗功能，同时还可以设置看门狗定时时间。

表 4.43 看门狗相关寄存器列表

寄存器名称	助记符	地址
32K 实时时钟配置寄存器	P_CLK_32K_CONF	0x88210114
WDOG 时钟配置寄存器	P_WDOG_CLK_CONF	0x88210084
WDOG 复位状态寄存器	P_WDOG_RESET_STATUS	0x882100E8
WDOG 控制寄存器	P_WDOG_MODE_CTRL	0x88170000
WDOG 复位周期设置寄存器	P_WDOG_CYCLE_SETUP	0x88170004
WDOG 清狗命令寄存器	P_WDOG_CLR_COMMAND	0x88170008

其中 32K 实时时钟配置寄存器在锁相环与时钟发生器章节已经介绍，这里只介绍剩余 5 个寄存器：

WDOG 时钟配置寄存器：P_WDOG_CLK_CONF(0x88210084)

通过 WDOG 时钟配置寄存器可以停止/打开 WDOG 时钟或者复位/不复位 WDOG 模块，在使能 WDOG 前需先打开 WDOG 时钟。

表 4.44 P_WDOG_CLK_CONF(0x88210084)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	WDOG_RST	WDOG_STOP

WDOG_RST　b1　看门狗模块时钟复位位：

0：看门狗模块时钟复位

1：看门狗模块时钟不复位

WDOG_STOP　b0　看门狗模块时钟使能位：

0：看门狗模块时钟停止

1：看门狗模块时钟使能

WDOG 复位状态寄存器：P_WDOG_RESET_STATUS(0x882100E8)

通过 WDOG 复位状态寄存器可以知道有没有发生看门狗复位。

表 4.45　P_WDOG_RESET_STATUS(0x882100E8)

位	b31~b3	b1	b0
读/写	-	R/W	R
默认值	-	0	0
名称	-	WDOG_INT	WDOG_ERRINT

WDOG_ INT　b1　看门狗错误复位标志位：

读 0：没有发生看门狗错误复位

读 1：发生看门狗错误复位

写 0：无意义

写 1：清除复位标志

WDOG_ ERRINT　b0　看门狗计数器溢出复位标志位：

0：没有发生看门狗计数器溢出复位

1：发生看门狗计数器溢出复位

WDOG 控制寄存器：P_WDOG_MODE_CTRL(0x88170000)

可以通过 WDOG 控制寄存器使能或者不使能看门狗功能。

表 4.46　P_WDOG_MODE_CTRL(0x88170000)

位	b31~b1	b0
读/写	-	R/W

默认值	-	0
名称	-	WDOG_EN

WDOG_EN b0 看门狗使能位：

0：不使能看门狗

1：使能看门狗

WDOG 复位周期设置寄存器：P_WDOG_CYCLE_SETUP(0x88170004)

SPCE3200 的看门狗模块有一个 32 位递减计数的计数器，可以通过 P_WDOG_CYCLE_SETUP 寄存器设置其计数初值，计数初值的设置范围为 0x00000001~0xFFFFFFFF。

表 4.47 P_WDOG_CYCLE_SETUP(0x88170004)

位	b31~b0
读/写	W
默认值	0x00000000
名称	RESET_COM

RESET_COM b31~b0 看门狗计数器计数初值设置位，设置范围：0x00000000~0xFFFFFFFF。

当通过 P_WDOG_CYCLE_SETUP 寄存器把看门狗计数器的计数初值设置为 RESET_COM 时，看门狗计数器的复位周期为 RESET_COM/32768（s）。

WDOG 清狗命令寄存器：P_WDOG_CLR_COMMAND(0x88170008)

当使能看门狗后，看门狗计数器按照 P_WDOG_CYCLE_SETUP 寄存器设置的计数初值开始递减计数，在还没有计数溢出前，如果通过 P_WDOG_CLR_COMMAND 寄存器清狗，看门狗计数器会重载计数初值并重新开始计数。

表 4.48 P_WDOG_CLR_COMMAND(0x88170008)

位	b31~b0
读/写	W
默认值	0
名称	CLEAR_COM

CLEAR_COM b31~b0 清狗写命令：

写 0xaxxxxxx5 清狗

写其他值系统复位

4.5.5 基本操作

使用看门狗功能通常有两个操作：使能看门狗和清看门狗。

使能看门狗的流程如图 4.23。

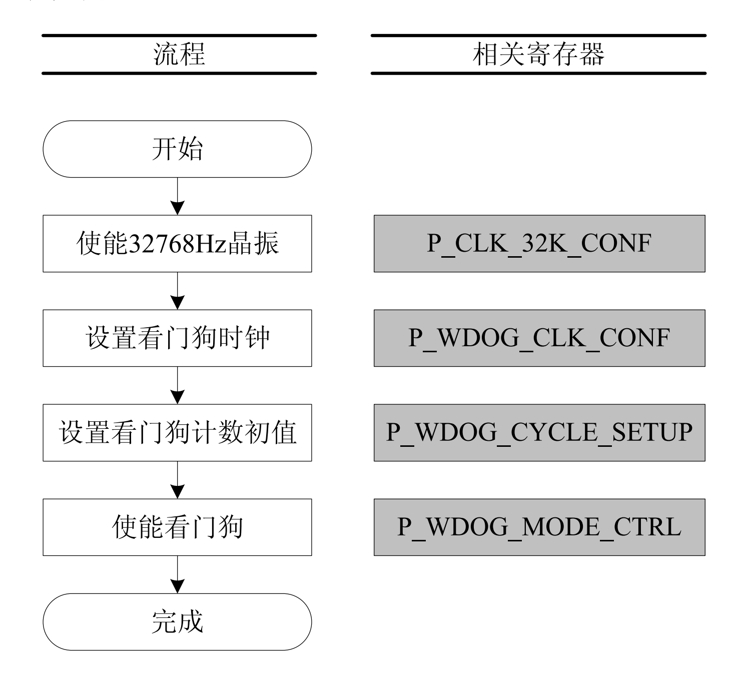


图 4.23 打开看门狗计数器

打开看门狗计数器后，需在计数器没有计数溢出前通过 P_WDOG_CLR_COMMAND 清狗，即向 P_WDOG_CLR_COMMAND 寄存器写 0xAxxxxxx5 即可。

以上两个操作可参考以下程序段：

```
*P_CLK_32K_CONF = C_32K_CRY_EN;                                  // 32768Hz 晶振打开
*P_WDOG_CLK_CONF = C_WDOG_CLK_EN | C_WDOG_RST_DIS; // watch dog 时钟配置
*P_WDOG_CYCLE_SETUP = 400000;                                    //  看门狗定时周期设置
*P_WDOG_MODE_CTRL = C_WDOG_CTRL_EN;                              //  使能看门狗功能

while(1)
{
    *P_WDOG_CLR_COMMAND = C_WDOG_CLR_COMMAND;    //  清看门狗
}
```

4.5.6 注意事项

在使用看门狗时需要注意以下几点：

■ 默认看门狗计数器是关闭的，即看门狗是禁止的，所以在使用前要先使能看门狗



- 在使用看门狗模块时，需要使能 32768Hz 晶振
- 清看门狗时注意写入寄存器 P_WDOG_CLR_COMMAND 的数据是 0xAxxxxxx5，如果写入其他数据会导致系统复位（看门狗错误复位）

4.6 睡眠与唤醒

4.6.1 睡眠

为了节省能源，SPCE3200 可工作在省电模式，也就是下文所说的睡眠；系统上电后，CPU 一直工作，直到检测到睡眠命令（sleep）；单片机接收到睡眠信号后关闭系统时钟（PLL），进入睡眠模式；进入睡眠模式后，程序计数器停在当前指令的下一条指令地址，如果有唤醒信号，系统从这个地址开始运行。

按照 PLL 是否被关闭，SPCE3200 有两种省电模式（睡眠模式）：

- Wait 模式：不关闭 PLL
- Halt 模式：关闭 PLL

4.6.2 睡眠相关寄存器

与睡眠相关的寄存器有两个：睡眠控制寄存器和睡眠时钟选择寄存器，如表 4.43。

表 4.49 睡眠相关寄存器列表

寄存器名称	助记符	地址
睡眠控制寄存器	P_SLEEP_MODE_CTRL	0x88210000
睡眠时钟选择寄存器	P_SLEEP_CLK_SEL	0x882100DC

睡眠模式控制寄存器：P_SLEEP_MODE_CTRL(0x88210000)

通过睡眠模式控制寄存器来控制系统进入 Wait 模式或者 Halt 模式。

表 4.50 P_SLEEP_MODE_CTRL(0x88210000)

位	b31~b7	b6	b5	b4	b3	b2	b1	b0
读/写	-	W	W	W	W	W	W	W
默认值	-	0	0	0	0	1	1	1
名称	-	CPUALLRST	CPUWRST	CPUPRST	CPUPLLIRQ	CPUCKMIU	CPUCKIRQ	CPUCK

CPUALLRST b6 暂时复位 CPU（到 warmrst_n 端口）和所有其它模块位：

0：不复位

1：复位

CPUWRST b5 暂时复位 CPU（到 warmrst_n 端口）位：

0：不复位

1：复位

CPUPRST　　b4　　暂时复位 CPU（到 Poweron_n 端口）位：

0：不复位

1：复位

CPUPLLIRQ　　b3　　停止 CPU 和 MIU 时钟然后停止 PLL 直到 IRQ 发生或者键状态改变位：

0：不停止

1：停止

CPUCKMIU　　b2　　停止 CPU 和 MIU 时钟直到 IRQ 发生或者键状态改变位：

0：停止

1：不停止

CPUCKIRQ　　b1　　停止 CPU 时钟直到 IRQ 发生或者键状态改变位：

0：停止

1：不停止

CPUCK　　b0　　暂时停止 CPU 时钟以改变 CPU 时钟频率位：

0：停止

1：不停止

说明：

1、通过 P_SLEEP_MODE_CTRL 的 b1 设置为 0 选择系统省电模式为 Wait 模式；

2、通过 P_SLEEP_MODE_CTRL 的 b2 设置为 0、b3 设置为 1 选择系统省电模式为 Halt 模式。

睡眠时钟选择寄存器：P_SLEEP_CLK_SEL (0x882100DC)

系统睡眠模式有两种时钟可以选择：PLLV/8 和 32768Hz，通过睡眠时钟选择寄存器可以选择睡眠模式利用哪种时钟。

表 4.51　P_SLEEP_CLK_SEL (0x882100DC)

位	b31~b1	b0
读/写	-	W
默认值	-	0
名称	-	SLEEP_CLK

SLEEP_CLK　　b0　　选择睡眠模式的时钟位

0：选择 PLLV/8 作为睡眠模式的时钟

1：选择 32768Hz 作为睡眠模式的时钟



4.6.3 唤醒

系统从睡眠模式被唤醒需要一个唤醒信号来接通系统时钟(PLL)电路，并通过产生唤醒 IRQ 中断信号引导 CPU 完成唤醒处理及系统初始化操作。唤醒中断操作完成后，程序计数器会指引 CPU 执行睡眠指令的下一条指令。

睡眠模式的唤醒源可来自键状态改变信号（键唤醒信号），也可以来自任一模块(该模块的时钟未被关闭)的中断信号（中断唤醒源）。

通过 P_WAKEUP_KEYC_SEL(0x88200008)寄存器来选择键唤醒的口，用来作为键唤醒信号的口分为 GROUP1、GROUP2、GROUP3、GROUP4、GROUP5 五组，如表 4.52。

表 4.52 键唤醒接口列表

序号	0x88200008[b2~b0]设置	组	接口	对应键唤醒口
1	001	GROUP1	LCD_D0	KEYCHANGE1_0
2			LCD_D1	KEYCHANGE1_1
3			LCD_D2	KEYCHANGE1_2
4			LCD_D3	KEYCHANGE1_3
5			LCD_D4	KEYCHANGE1_4
6			LCD_D5	KEYCHANGE1_5
7			LCD_D6	KEYCHANGE1_6
8			LCD_D7	KEYCHANGE1_7
9			LCD_D8	KEYCHANGE1_8
10			LCD_D9	KEYCHANGE1_9
11			LCD_D10	KEYCHANGE1_10
12			LCD_D11	KEYCHANGE1_11
13			LCD_D12	KEYCHANGE1_12
14			LCD_D13	KEYCHANGE1_13
15			LCD_D14	KEYCHANGE1_14
16			LCD_D15	KEYCHANGE1_15
17	010	GROUP2	CSI_D0	KEYCHANGE2_0
18			CSI_D1	KEYCHANGE2_1
19			CSI_D2	KEYCHANGE2_2
20			CSI_D3	KEYCHANGE2_3
21			CSI_D4	KEYCHANGE2_4
22			CSI_D5	KEYCHANGE2_5

序号	0x88200008[b2~b0]设置	组	接口	对应键唤醒口
23			CSI_D6	KEYCHANGE2_6
24			CSI_D7	KEYCHANGE2_7
25			CSI_CKO	KEYCHANGE2_8
26			CSI_CKI	KEYCHANGE2_9
27			CSI_HS	KEYCHANGE2_10
28			CSI_VS	KEYCHANGE2_11
29			CSI_FIELD	KEYCHANGE2_12
30			I2C_CLK	KEYCHANGE2_13
31			I2C_DATA	KEYCHANGE2_14
32	011	GROUP3	CSI_HS	KEYCHANGE3_0
33			CSI_VS	KEYCHANGE3_1
34			CSI_FIELD	KEYCHANGE3_2
35			UART_RX	KEYCHANGE3_3
36			UART_TX	KEYCHANGE3_4
37			ADC_CH4	KEYCHANGE3_5
38			ADC_CH5	KEYCHANGE3_6
39			ADC_CH6	KEYCHANGE3_7
40			ADC_CH7	KEYCHANGE3_8
41	100	GROUP4	NFLASH_D6	KEYCHANGE4_0
42			NFLASH_REN	KEYCHANGE4_1
43			NFLASH_WEN	KEYCHANGE4_2
44			NFLASH_CEN	KEYCHANGE4_3
45			NFLASH_RDY	KEYCHANGE4_4
46			ADC_CH4	KEYCHANGE4_5
47			ADC_CH5	KEYCHANGE4_6
48			ADC_CH6	KEYCHANGE4_7
49			ADC_CH7	KEYCHANGE4_8
50			NFLASH_D7	KEYCHANGE4_9
51	101	GROUP5	NFLASH_ALE	KEYCHANGE2_0



序号	0x88200008[b2~b0]设置	组	接口	对应键唤醒口
52			NFLASH_REN	KEYCHANGE2_1
53			NFLASH_WEN	KEYCHANGE2_2
54			NFLASH_CEN	KEYCHANGE2_3
55			NFLASH_RDY	KEYCHANGE2_4
56			NFLASH_D4	KEYCHANGE2_5
57			NFLASH_D5	KEYCHANGE2_6
58			NFLASH_D6	KEYCHANGE2_7
59			NFLASH_D7	KEYCHANGE2_8
60			SPI_CSN	KEYCHANGE2_9
61			CSI_HS	KEYCHANGE2_10
62			CSI_VS	KEYCHANGE2_11
63			CSI_FIELD	KEYCHANGE2_12

用来作为中断唤醒信号的中断唤醒源如表 4.53。

表 4.53 睡眠模式的中断唤醒源

唤醒中断源	说明	唤醒中断源	说明
Mic 中断	通过 Mic 引入的中断	Flash 中断	Nand Flash 中断
ADC 中断	通过普通 A/D 通道引入的中断	SD 中断	SD 卡中断
时基中断	时基中断	I2C 中断	I2C 主机中断
计数器中断	Timer0~5 引入的中断	I2S 中断	I2S 主/从中断
TV 中断 1	TV 垂直消隐启动中断	APBDMA 中断 1	APBDMA 通道 1 操作完成中断
LCD 中断	LCD 垂直消隐启动中断	APBDMA 中断 2	APBDMA 通道 2 操作完成中断
TV 中断 3	TV 光枪击中中断	LDMDMA 中断	LDM DMA 操作完成中断
CSI 中断 1	传感器帧结束中断	BLN 中断	混合 DMA 操作完成中断
CSI 中断 3	传感器坐标击中中断	APBDMA 中断 3	APBDMA 通道 3 操作完成中断
CSI 中断 2	传感器运动帧结束中断	APBDMA 中断 4	APBDMA 通道 4 操作完成中断
CSI 中断 0	捕获完成中断	RTC 中断	实时时钟中断
TV 中断 4	TV 坐标击中中断	MPG 中断	MPG4 一帧编/解码完成中断
USB 中断	USB 主/从设备中断	ECC 中断	ECC 发现错误的事件中断

唤醒中断源	说明	唤醒中断源	说明
SIO 中断	SIO 中断	SFTCFG 中断	指定管脚的上升沿/下降沿
SPI 中断	SPI 中断	键值改变中断	键值改变中断
UART 中断	UART 中断	LVD 中断	低电压检测中断

4.6.4 键唤醒相关寄存器

键唤醒有两个控制寄存器：键唤醒控制寄存器和键唤醒清除寄存器，如表 4.54。

表 4.54 键唤醒相关寄存器列表

寄存器中文名称	寄存器英文名称	地址
键唤醒选择寄存器	P_WAKEUP_KEYC_SEL	0x88200008
键唤醒清除寄存器	P_WAKEUP_KEYC_CLR	0x882100C0

键唤醒控制寄存器：P_WAKEUP_KEYC_SEL(0x88200008)

可以通过 P_WAKEUP_KEYC_SEL 的 b2~b0 选择 GROUP1、GROUP2、GROUP3、GROUP4、GROUP5 五组中任一组作为键唤醒口。

表 4.55 P_WAKEUP_KEYC_SEL(0x88200008)

位	b31~b3	b2~b0
读/写	-	R/W
默认值	-	000
名称	-	SW_KEYC

SW_KEYC b1~b0 选择键唤醒接口，参考表 4.52：

001：选择 GROUP1 组接口为键唤醒口

010：选择 GROUP2 组接口为键唤醒口

011：选择 GROUP3 组接口为键唤醒口

100：选择 GROUP4 组接口为键唤醒口

101：选择 GROUP5 组接口为键唤醒口

其他：不选择键唤醒口

键唤醒清除寄存器：P_WAKEUP_KEYC_CLR(0x882100C0)

键唤醒清除寄存器用来清除键唤醒。当一键唤醒事件发生时，必须通过该寄存器清除键唤醒。

表 4.56 P_WAKEUP_KEYC_CLR(0x882100C0)

位	b31~b1	b0

读/写	-	R/W
默认值	-	0
名称	-	KEYCHG_CLR

KEYCHG_CLR b0 键唤醒清除位：

0：清除键唤醒

1：使能键唤醒

4.6.5 应用举例

【例 4.7】：

利用 Timer 中断实现 1 个发光二极管电平翻转（即闪烁），在两次发光二极管电平翻转之间的时间间隔里使系统进入 Wait 节电模式，直到 Timer 中断。电路连接图如图 4.24 所示：

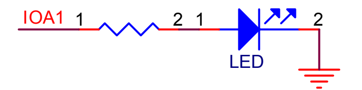


图 4.24 电路连接图

中断唤醒的一般操作流程如图 4.25，步骤如下：

1、 打开中断；

2、 使能模块时钟；

3、 使能模块；

4、 使能模块中断，清中断标志；

5、 选择睡眠模式，给睡眠指令，此时系统进入睡眠模式；注意图中的等待唤醒中断信号由硬件操作，同样软件不需要做任何操作。

6、 唤醒后进入中断服务程序，进行中断服务处理。



图 4.25　中断唤醒的操作流程图

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
        *P_INT_MASK_CTRL1 &= ~C_INT_TIMER_DIS;               // 打开 TIMER 中断
        *P_IOA_GPIO_SETUP = 0x00000202;                      // 使能 IOA1
        *P_TIMER0_CLK_CONF = 0;                              // 复位 TIMER0
        *P_TIMER0_CLK_CONF = C_TIMER_CLK_EN
                        | C_TIMER_RST_DIS;                   // 使能 TIMER0
        *P_TIMER_CLK_SEL = C_TIMER0_CLK_27M | 247;           // 选择 TIMER0 时钟，分频 248
        *P_TIMER0_PRELOAD_DATA = 0x80;                       // 载入计数初值
        *P_TIMER0_CCP_CTRL = C_TIMER_NOR_MODE;               // 选择定时工作模式
        *P_TIMER0_MODE_CTRL = C_TIMER_CTRL_EN
                        | C_TIMER_INT_EN | C_TIMER_INT_FLAG; // 使能中断
```

```
        while(1)
        {
            *P_SLEEP_MODE_CTRL = C_SLEEP_WAIT_MODE;     // 关闭 CPU 时钟，wait 模式
            __asm("sleep");                             // 进入睡眠模式
        }
        return 0;
    }
    //=========================================================
    // 语法格式：void IRQ56(void)
    // 功能描述: 定时器（Timer）中断服务函数
    // 入口参数: 无
    // 出口参数: 无
    //=========================================================
    void IRQ56(void)
    {
        *P_TIMER0_MODE_CTRL |= C_TIMER_INT_FLAG;    // 清除中断
        *P_IOA_GPIO_SETUP ^= 0x00000002;            // IOA 输出反相
    }
```

4.7 ADC

4.7.1 概述

SPCE3200 具有 1 个 12 位 9 路的 ADC（A/D Converter），该 ADC 不仅具有普通 A/D 转换功能，而且具有语音录制专用 A/D 转换功能，其中：有 1 路 MIC 专用输入通道，8 路通用 A/D 输入通道。

当 ADC 作为通用 A/D 使用时，支持可编程采样频率的自动采样方式，同时也支持通过采样命令控制的手动采样方式；所有 8 个普通 A/D 输入通道都通过中断读取转换数据。

MIC 输入通道支持 PGA，通过设置寄存器控制其增益；不同于通用 A/D 转换通道，MIC 模式下仅能通过 DMA 传输数据。

SPCE3200 的 ADC 时钟源来自 PLLA 和 PLLU，经过分频后作为 ADC 的时钟，无论是时钟源还是分频倍数都可以通过软件编程选择。

4.7.2 特性

SPCE3200 的 ADC 具有如下特性：

- 1 个 12 位 9 通道模数转换器
- 具有通用 A/D 和音频双重功能
- 测量电压：0~VRT（参考电压）
- 片上集成前级放大器和 MIC 增益可编程放大器

- 片上集成抗混淆滤波器和 MIC 偏压输出模块
- 集成参考电压模块和偏压电流/电压产生器
- 掉电模式消耗电流：1uA
- 通用 A/D 消耗电流：2.0mA
- 音频 MIC 消耗电流：4.0mA（不带 MIC 偏压输出），7.5mA（带 MIC 偏压输出）

4.7.3 引脚描述

SPCE3200 的 A/D 引脚如表 4.57。

表 4.57 SPCE3200 的 A/D 引脚列表

引脚名称	引脚号	引脚属性	功能描述
ADC_CH0	160	I	ADC 通道 0 模拟输入脚
ADC_CH1	161	I	ADC 通道 1 模拟输入脚
ADC_CH2	162	I	ADC 通道 2 模拟输入脚
ADC_CH3	163	I	ADC 通道 3 模拟输入脚
ADC_CH4	164	I	ADC 通道 4 模拟输入脚
ADC_CH5	165	I	ADC 通道 5 模拟输入脚
ADC_CH6	166	I	ADC 通道 6 模拟输入脚
ADC_CH7	167	I	ADC 通道 7 模拟输入脚
ADC_ADVRT	169	I	ADC 参考电压输入脚
ADC_ADFLT	172	O	反锯齿波输出脚
ADC_VREF	173	O	AFE 参考电压输出脚
ADC_MICIN	174	I	MIC 输入口
ADC_MICBIAS	175	O	缓存适配电压为 MIC 提供偏压。电压为 AVDD33 的 3/4
ADC_TESTN	176	I/O	测试模式的负输入或输出
ADC_TESTP	177	I/O	测试模式的正输入或输出

4.7.4 结构框图

SPCE3200 ADC 的结构框图如图 4.26。SPCE3200 的 ADC 时钟源来自 PLLA 和 PLLU，经过分频后作为 ADC 的时钟；9 个 ADC 通道可编程选择；A/D 转换控制器控制 A/D 转换；转换完成后通用 A/D 通道的转换数据通过中断控制读取，而 MIC 通道的转换数据只能通过 DMA 读取。

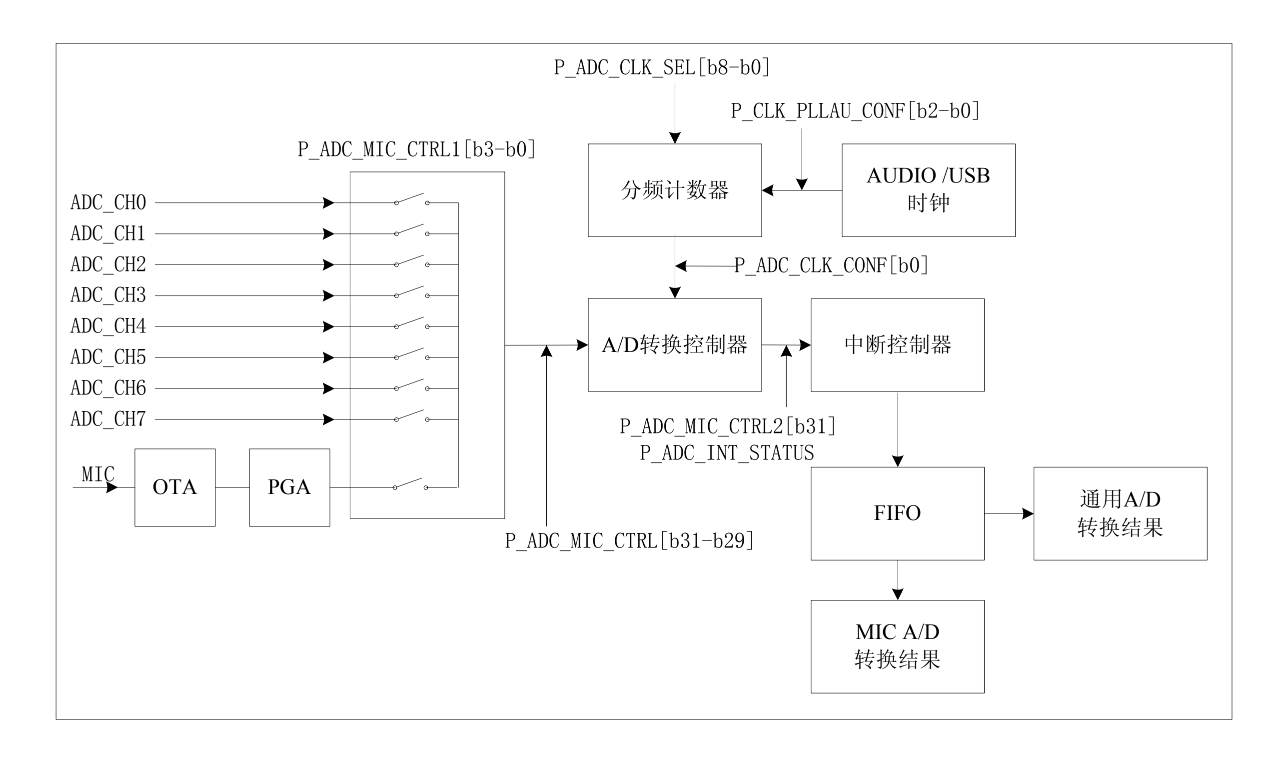


图 4.26 SPCE3200 ADC 的结构框图

4.7.5 寄存器描述

SPCE3200 的 ADC 相关寄存器共有 16 个，如表 4.58。

表 4.58 ADC 相关寄存器列表

寄存器名称	助记符	地址
ADC GPIO 设置寄存器	P_ADC_GPIO_SETUP	0x88200048
ADC GPIO 输入数据寄存器	P_ADC_GPIO_INPUT	0x88200078
ADC GPIO 外部中断寄存器	P_ADC_GPIO_INT	0x8820009C
ADC 模拟输入控制寄存器	P_ADC_AINPUT_CTRL	0x88200054
PLLAU 时钟配置寄存器	P_CLK_PLLAU_CONF	0x882100BC
ADC 时钟配置寄存器	P_ADC_CLK_CONF	0x882100AC
ADC 时钟选择寄存器	P_ADC_CLK_SEL	0x882100B0
ADC 控制寄存器 1	P_ADC_MIC_CTRL1	0x881A0000
MIC 增益寄存器	P_ADC_MIC_GAIN	0x881A0004
ADC 采样时钟寄存器	P_ADC_SAMPLE_CLK	0x881A0008
ADC 采样保持寄存器	P_ADC_SAMPLE_HOLD	0x881A000C
ADC 控制寄存器 2	P_ADC_MIC_CTRL2	0x881A0010
ADC 中断状态寄存器	P_ADC_INT_STATUS	0x881A0014

寄存器名称	助记符	地址
ADC 手动方式数据寄存器	P_ADC_MANUAL_DATA	0x881A0018
ADC 自动方式数据寄存器	P_ADC_AUTO_DATA	0x881A001C
MIC 转换数据寄存器	P_ADC_MIC_DATA	0x881A0020

如果 ADC 的模拟输入口复用为 GPIO 功能，可以对表 4.58 中前三个寄存器操作。寄存器的各位对应引脚关系如表 4.142：

表 4.59　ADC 模拟输入口复用为 GPIO 引脚与寄存器位对应关系

引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
ADC_CH0	160	bit[0]	可以
ADC_CH1	161	bit[1]	可以
ADC_CH2	162	bit[2]	可以
ADC_CH3	163	bit[3]	可以
ADC_CH4	164	bit[4]	不可以
ADC_CH5	165	bit[5]	不可以
ADC_CH6	166	bit[6]	不可以
ADC_CH7	167	bit[7]	不可以

ADC GPIO 设置寄存器：P_ADC_GPIO_SETUP(0x88200048)

ADC_CH0~ADC_CH7 除可以用来作为专用的 ADC 模拟信号输入口外，还可以用来做 GPIO。该寄存器控制 ADC_CH0~ADC_CH7 作为 GPIO 时的输出使能、上拉电阻输入/下拉电阻输入设置和输出数据。

表 4.60　P_ADC_GPIO_SETUP(0x88200048)

位	b31~b24	b23~b16	b15~b8	b7~b0
读/写	R/W	R/W	R/W	R/W
默认值	0xFF	0xFF	0x00	0x00
名称	ADC_PD	ADC_PU	ADC_OE	ADC_O

ADC_PD　b31~b24　ADC GPIO 下拉电阻输入口使能位：

0：不使能为下拉电阻输入口

1：使能为下拉电阻输入口

ADC_PU　b23~b16　ADC GPIO 上拉电阻输入口使能位：

0：使能为上拉电阻输入口



1：不使能为上拉电阻输入口

ADC_OE　b15~b8　ADC GPIO 输出使能位：

0：不使能为输出口

1：使能为输出口

ADC_O　b7~b0　ADC GPIO 输出数据

说明：寄存器 P_ADC_GPIO_SETUP 的 bit0、bit8、bit16、bit24 控制 ADC_CH0，按顺序 bit7、bit15、bit23、bit31 控制 ADC_CH7。

ADC GPIO 输入数据寄存器 P_ADC_GPIO_INPUT(0x88200078)

ADC_CH0~ADC_CH7 作为通用输入口时的输入数据寄存器。

表 4.61　P_ADC_GPIO_INPUT(0x88200078)

位	b31~b8	b7~b0
读/写	-	R
默认值	-	0x00
名称	-	ADC_INPUT

ADC_INPUT　b7~b0　输入数据

说明：b0 对应 ADC_CH0 输入数据，b7 对应 ADC_CH7 输入数据。

ADC GPIO 外部中断寄存器：P_ADC_GPIO_INT(0x8820009C)

ADC_CH0~ADC_CH3 作为外部中断输入时的控制寄存器，控制外部中断的使能和中断标志位的清除。

表 4.62　P_ADC_GPIO_INT(0x8820009C)

位	b31~b28	b27~b24	b23~b20	b19~b16	b15~b12	b11~b8	b7~b4	b3~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W
默认值	-	0	-	0	-	0	-	0
名称	-	ADC_FI	-	ADC_RI	-	ADC_FIEN	-	ADC_RIEN

ADC_FI　b27~b24　ADC GPIO 下降沿中断事件标志位：

读 0：没有发生下降沿中断

读 1：发生下降沿中断

写 0：无意义

写 1：清除中断标志

ADC_RI　　b19~b16　ADC GPIO 上升沿中断事件标志位：

读 0：没有发生上升沿中断

读 1：发生上升沿中断

写 0：无意义

写 1：清除中断标志

ADC_FIEN　b11~b8　ADC GPIO 下降沿中断使能位：

0：不使能下降沿中断

1：使能下降沿中断

ADC_RIEN　b3~b0　ADC GPIO 上升沿中断使能位：

0：不使能上升沿中断

1：使能上升沿中断

注意：只有 ADC_CH0~ ADC_CH3 可以作为外部中断的输入口，详细可参考 GPIO 章节。

ADC 模拟输入控制寄存器：P_ADC_AINPUT_CTRL(0x88200054)

ADC 的接口 ADC_CH0~ADC_CH7 作为模拟信号输入口时，需要先通过 P_ADC_AINPUT_CTRL 寄存器使能。

表 4.63　P_ADC_AINPUT_CTRL(0x88200054)

位	b31~b8	b7~b0
读/写	-	R/W
默认值	-	0x00
名称	-	ADC_GPIO_AEN

ADC_GPIO_AEN　b7~b0　ADC 模拟输入口使能位：

0：不使能为模拟输入口

1：使能为模拟输入口

说明：b0 对应 ADC_CH0，b7 对应 ADC_CH7。

PLLAU 设置寄存器：P_CLK_PLLAU_CONF(0x882100BC)

P_CLK_PLLAU_CONF 寄存器的介绍请参考锁相环与时钟发生器章节。

ADC 时钟配置寄存器：P_ADC_CLK_CONF(0x882100AC)

通过 ADC 时钟配置寄存器可以停止/打开 ADC 时钟或者复位/不复位 ADC 模块，在使能 ADC

前需先打开 ADC 时钟。

表 4.64 P_ADC_CLK_CONF(0x882100AC)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	ADC_RST	ADC_STOP

ADC_RST b1 ADC 模块时钟复位位：

0：ADC 模块时钟复位

1：ADC 模块时钟不复位

ADC_STOP b0 ADC 模块时钟使能位：

0：ADC 模块时钟停止

1：ADC 模块时钟使能

ADC 时钟选择寄存器：P_ADC_CLK_SEL(0x882100B0)

SPCE3200 ADC 的时钟由 PLLA 或者 PLLU 两种时钟源可供选择，通过 P_ADC_CLK_SEL 寄存器来选择 PLLA 作为 ADC 的时钟源，还是 PLLU 作为 ADC 的时钟源。

表 4.65 P_ADC_CLK_SEL(0x882100B0)

位	b31~b29	b8	b7~b0
读/写	-	W	W
默认值	-	0	0x01
名称	-	ADC_Source	DividedNUM

ADC_Source b8 ADC 时钟源选择位：

0：选择 PLLA 作为 ADC 的时钟源

1：选择 PLLU 作为 ADC 的时钟源

DividedNUM b7~b0 ADC 时钟设置位，ADC 时钟为时钟源的（DividedNUM+1）分频。

DividedNUM!=0

ADC 控制寄存器 1：P_ADC_MIC_CTRL1(0x881A0000)

SPCE3200 的 ADC 有 9 个通道，两种功能：通用 ADC 和音频专用 ADC（即 MIC）；通过 ADC 控制寄存器可以使能通用 ADC 或者 MIC、选择 A/D 通道。

表 4.66 P_ADC_MIC_CTRL1(0x881A0000)

位	b31	b30	b29	b27	b26	b25~b4	b3~b0

读/写	R/W	R/W	R/W	R/W	R/W	-	R/W
默认值	0	0	0	0	0	-	0
名称	ADC_SEL	ADC_EN	MIC_EN	MIC_BOOST	MIC_BIAS	-	CH_SEL

ADC_SEL　　b31　　通用 ADC 选择位：

0：不选择通用 ADC

1：选择通用 ADC

ADC_EN　　b30　　通用 ADC 使能位：

0：不使能通用 ADC

1：使能通用 ADC

MIC_EN　　b29　　MIC 使能位：

0：不使能 MIC

1：使能 MIC

MIC_BOOST　　b27　　MIC 前级放大器使能位（仅对 MIC 模式有用）：

0：不使能 MIC 前级放大器

1：使能 MIC 前级放大器

MIC_BIAS　　b26　　MIC 偏压使能位：

0：不使能 MIC 偏压

1：使能 MIC 偏压

CH_SEL　　b3~b0　　通道选择位：

0000：选择通用 A/D 通道 0(ADC_CH0)

0001：选择通用 A/D 通道 1(ADC_CH1)

0010：选择通用 A/D 通道 2(ADC_CH2)

0011：选择通用 A/D 通道 3(ADC_CH3)

0100：选择通用 A/D 通道 4(ADC_CH4)

0101：选择通用 A/D 通道 5(ADC_CH5)

0110：选择通用 A/D 通道 6(ADC_CH6)

0111：选择通用 A/D 通道 7(ADC_CH7)

1xxx：选择 MIC 通道

MIC 增益寄存器：P_ADC_MIC_GAIN(0x881A0004)

SPCE3200 ADC 的 MIC 通道支持 PGA，通过 MIC 增益寄存器来选择设置其增益。

表 4.67　P_ADC_MIC_GAIN(0x881A0004)

位	b31~b5	b4~b0

读/写	-	R/W
默认值	-	0
名称	-	MIC_GAIN

MIC_GAIN b4~b0 MIC 增益值选择：

00000：33.0db

00001：31.5db

00010：30.0db

00011：28.5db

……

10101：1.5db

10110：0.0db

10111：-1.5db

……

11110：-12.0db

11111：∞

ADC 采样时钟寄存器 P_ADC_SAMPLE_CLK(0x881A0008)

不同于其它的 ADC，SPCE3200 的 ADC 可通过 P_ADC_SAMPLE 编程设置其时钟周期，和 ADC 采样保持寄存器结合设置其采样率。

表 4.68 P_ADC_SAMPLE_CLK(0x881A0008)

位	b31~b24	b15~b0
读/写	-	R/W
默认值	-	0x0000
名称	-	ADC_CLOCK_CYCLE

ADC_CLOCK_CYCLE b15~b8 ADC 时钟周期 ＝ADC_CLOCK_CYCLE+1

注意：只有自动采样方式时需要设置其时钟采样周期。

ADC 采样保持寄存器：P_ADC_SAMPLE_HOLD(0x881A000C)

SPCE3200 的 ADC 作通用 A/D 使用时，支持可编程采样频率的自动采样方式，同时也支持通过采样命令控制的手动采样方式；通过 P_ADC_SAMPLE_HOLD 可设置采样频率。

表 4.69 P_ADC_SAMPLE_HOLD(0x881A000C)

位	b31~b28	b27~b16	b15~b8
读/写	W	-	R/W
默认值	0	-	0x00
名称	HOLD_WIDTH	-	HOLD_CYLCE

HOLD_WIDTH　b31~b28　采样保持带宽设置：HOLD_WIDTH>=2

HOLD_CYLCE　b15~b0　采样保持周期设置：HOLD_CYLCE >=15

说明：使能 ADC 时钟之后，系统会自动把 P_ADC_SAMPLE_HOLD 设置为 0x200f，即 HOLD_CYLCE 为 15，HOLD_WIDTH 为 2。

注意：只有自动采样方式时需要设置其时钟采样周期。关于设置 ADC 时钟和采样保持宽度/周期数的方法举例来说，假设需要 44.1KHz 的 ADC 采样率，且 ADC 控制器时钟为 33.8688 MHz，则采样周期数即为 33.8688M / 44.1K = 768，且若需采样保持周期数为 16，则最终：

ADC 时钟周期数 = 768 / 16 = 48；

因此，ACC = 47，采样保持周期 HOLD_CYCLE = 15，采样保持宽度 HOLD_WIDTH = 2

ADC 控制寄存器 2：P_ADC_MIC_CTRL2(0x881A0010)

通过 ADC 控制寄存器 2 可以使能 ADC 控制器、选择通用 A/D 的采样方式等。

表 4.70　P_ADC_MIC_CTRL2(0x881A0010)

位	b31	b27	b26	b25
读/写	R/W	R/W	R/W	R/W
默认值	0	0	0	0
名称	ADC_EN	AUTO_SAMPLE	MIC_MODE	MUTE_SEL
位	b24	b23	b6~b4	b2~b0
读/写	R/W	R/W	R/W	R/W
默认值	0	0	0	0
名称	DMA_SIZE	AUTO_CLR	FIFO_INT	FIFO_RD

ADC_EN　b31　ADC 控制器使能位：

0：不使能 ADC 控制器

1：使能 ADC 控制器

AUTO_SAMPLE　b27　通用 A/D 采样方式选择：

0：选择手动采样方式

1：选择自动采样方式



MIC_MODE	b26	MIC 模式数据类型选择（无符号数/有符号数）： 0：有符号数 1：无符号数
MUTE_SEL	b25	MIC 模式的静音选择位： 0：正常选择 1：选择静音
DMA_SIZE	b24	MIC 模式 DMA 数据大小的选择： 0：32 位 1：16 位
AUTO_CLR	b23	通用 A/D 模式自动清除中断标志使能位： 0：不使能通用 A/D 模式自动清除中断标志 1：使能通用 A/D 模式自动清除中断标志
FIFO_INT	b6~b4	中断时 FIFO 接收的数据量设置： 000：接收到 1 个数据时中断 001：接收到 2 个数据时中断 010：接收到 3 个数据时中断 011：接收到 4 个数据时中断 100：接收到 5 个数据时中断 101：接收到 6 个数据时中断 110：接收到 7 个数据时中断 111：接收到 8 个数据时中断
FIFO_RD	b[2:0]	FIFO 读接收数据量设置

ADC 中断状态寄存器：P_ADC_INT_STATUS(0x881A0014)

通过 ADC 中断状态寄存器可以使能通用 ADC 中断、MIC 溢出中断、FIFO 接收溢出中断等。

表 4.71 P_ADC_INT_STATUS(0x881A0014)

位	b31	b30	b29	b28	b27
读/写	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0
名称	ADC_INT	ADC_INTEN	ADC_AINT	ADC_AINTEN	MIC_OFINT
位	b26	b23	b22	b21	b15
读/写	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0
名称	MIC_OFINTEN	FIFO_OFE	FIFO_FULL	FIFO_EMPTY	CONVERT_ADC

ADC_INT	b31	通用 A/D 手动采样方式中断标志位： 读 0：发生通用 A/D 手动采样方式中断 读 1：没有发生通用 A/D 手动采样方式中断 写 0：无意义 写 1：清中断标志
ADC_INTEN	b30	通用 A/D 手动采样方式中断使能位： 0：不使能通用 A/D 手动采样方式中断 1：使能通用 A/D 手动采样方式中断
ADC_AINT	b29	通用 A/D 自动采样方式中断标志位： 读 0：发生通用 A/D 自动采样方式中断 读 1：没有发生通用 A/D 自动采样方式中断 写 0：无意义 写 1：清中断标志
ADC_AINTEN	b28	通用 A/D 自动采样方式中断使能位： 0：不使能通用 A/D 自动采样方式中断 1：使能通用 A/D 自动采样方式中断
MIC_OFINT	b27	MIC 溢出中断标志位： 读 0：发生 MIC 溢出中断 读 1：没有发生 MIC 溢出中断 写 0：无意义 写 1：清中断标志
MIC_OFINTEN	b26	MIC 溢出中断使能位： 0：不使能 MIC 溢出中断 1：使能 MIC 溢出中断
FIFO_OFE	b23	FIFO 接收溢出错误标志位： 读 0：发生溢出错误 读 1：没有发生溢出错误 写 0：无意义 写 1：清溢出错误标志
FIFO_FULL	b22	FIFO 满标志： 0：非满 1：满
FIFO_EMPTY	b21	FIFO 空标志： 0：非空



1：空

CONVERT_ADC　b15　　通用 A/D 手动方式启动转换位：

0：不启动手动转换

1：启动手动转换

ADC 手动方式数据寄存器：P_ADC_MANUAL_DATA(0x881A0018)

P_ADC_MANUAL_DATA 寄存器用来保存通用 A/D 手动采样方式转换的数据，结果保存在寄存器的高 12 位。

表 4.72　P_ADC_MANUAL_DATA(0x881A0018)

位	b31~b20	b19~b0
读/写	R	-
默认值	0	-
名称	ADC_SELF_DATA	-

ADC_SELF_DATA　b31~b20　　通用 A/D 手动采样方式转换数据，保存在寄存器的高 12 位

ADC 自动方式数据寄存器：P_ADC_AUTO_DATA(0x881A001C)

P_ADC_AUTO_DATA 寄存器用来保存通用 A/D 自动采样方式转换的数据，结果保存在寄存器的高 12 位。

表 4.73　P_ADC_AUTO_DATA(0x881A001C)

位	b31~b20	b19~b0
读/写	R	-
默认值	0	-
名称	ADC_AUTO_DATA	-

ADC_AUTO_DATA　b31~b20　　通用 A/D 自动采样方式转换数据，保存在寄存器的高 12 位

MIC 转换数据寄存器：P_ADC_MIC_DATA(0x881A0020)

P_ADC_MIC_DATA 寄存器用来保存通过 MIC 通道输入 A/D 转换的数据。其中低字节保存 1 个 16 位的 MIC 转换数据，高字节保存 1 个 16 位的 MIC 转换数据。2 个数据按照采样频率转换存储。其中转换的数据是 12 位的，经过插值算法扩展为 16 位的数据。

表 4.74　P_ADC_MIC_DATA(0x881A0020)

位	b31~b0
读/写	R

默认值	0x00000000
名称	MIC_DATA

MIC_DATA b31~b0 MIC 通道输入 A/D 转换的数据

4.7.6 基本操作

ADC 的基本操作主要包括 ADC 时钟的设置和 ADC 的启动转换，同时会介绍通用 ADC 的中断。

1、 ADC 时钟的设置

SPCE3200 ADC 的时钟源有两种：PLLA 或者 PLLU，其时钟是由这两种时钟源经过分频提供的，分频倍数可软件设置。

SPCE3200 ADC 时钟设置流程如图 4.27。

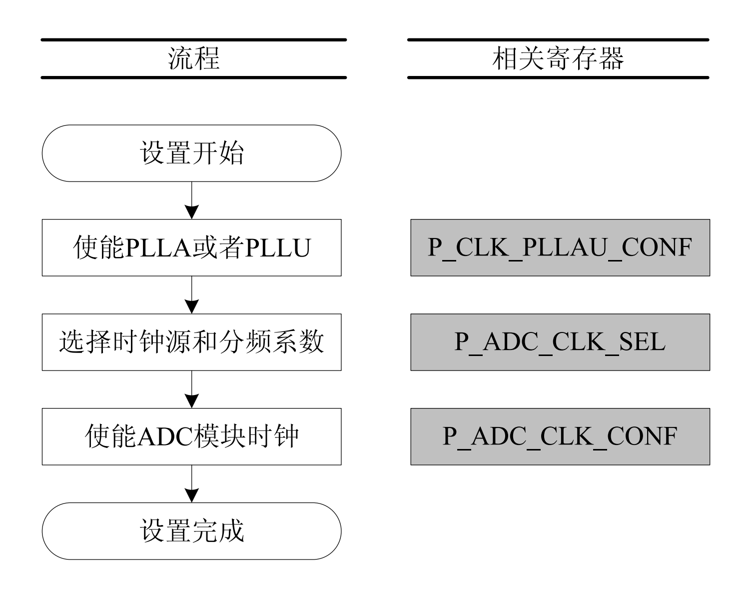


图 4.27 SPCE3200 ADC 时钟设置流程图

程序段参考如下：

```
    *P_ADC_CLK_CONF = C_ADC_CLK_EN
                    | C_ADC_RST_DIS;        // ADC 时钟模块设置
    *P_CLK_PLLAU_CONF = C_PLLA_CLK_EN  //  使能 PLLA
                    | C_PLLA_CLK_67M;      //  频率选择为 67MHz
    *P_ADC_CLK_SEL = C_ADC_SEL_PLLA        // ADC 时钟源选择 PLLA
                    | 0x01;                 // ADC 时钟频率选择为 PLLA 的 2 分频
```

2、 ADC 的启动

SPCE3200 的 ADC 有两种功能：通用 A/D 和专用 MIC 输入 A/D。

通用 A/D 包括自动采样方式和手动采样方式。



通用 A/D 的自动采样方式启动流程如图 4.28。步骤如下：

（1） 使能 A/D 的数据口为模拟输入口；

（2） 设置为通用 A/D 模式，选择 A/D 通道；

（3） 设置采样保持周期及带宽；

（4） 使能 ADC 中断；

（5） 使能 ADC 控制器并选择为自动采样方式。



图 4.28 通用 A/D 的自动采样方式启动流程图

程序段参考如下：

```
*P_ADC_AINPUT_CTRL = C_ADC_CH0_EN;          // ADC 的 channal 0 模拟输入使能
*P_ADC_MIC_CTRL1 = C_ADC_MODE_ADC           // 选择 ADC 模式
                   | C_ADC_CH0_SEL          // 选择 channal 0
                   ;
*P_ADC_SAMPLE_CLK = 0x00000017;             // 设定 ADC clock cycle 为 24 即(23+1)
*P_ADC_SAMPLE_HOLD = 0x2000000f;            // 设定采样保持 cycle 为 16,带宽为 2
*P_ADC_INT_STATUS = C_ADC_AUTO_INTEN        // 使能 ADC 自动方式的中断
                   | C_ADC_AUTO_FLAG;       // 清中断标志
*P_ADC_MIC_CTRL2 = C_ADC_CTRL_EN            // 使能 ADC 控制器
                   | C_ADC_AUTO_MODE        // 选择自动采样模式
                   | C_ADC_AUTO_CLR         // 自动清除中断标志
                   ;
```

通用 A/D 的手动采样方式启动流程如图 4.29。注意手动采样方式不需要设置 ACC 和采样保持周期及带宽；但是要启动手动转换后才开始一次 A/D 转换。

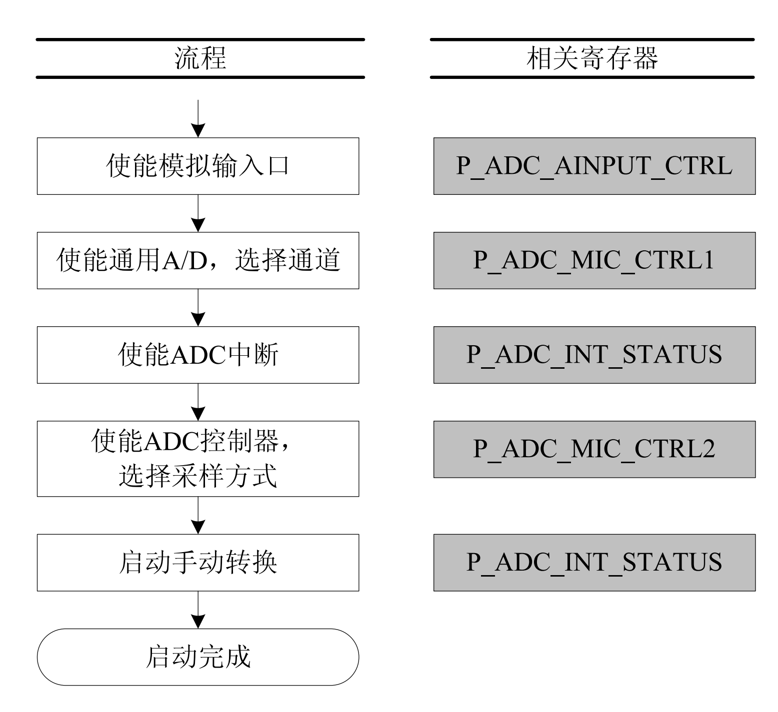


图 4.29　通用 A/D 的手动采样方式启动流程图

程序段参考如下：

```
*P_ADC_AINPUT_CTRL = C_ADC_CH0_EN;              // ADC 的 channal 0 模拟输入使能
*P_ADC_MIC_CTRL1 =      C_ADC_MODE_ADC          // 选择 ADC 模式
                        | C_ADC_CH0_SEL         // 选择 channal 0
                          ;
*P_ADC_INT_STATUS = C_ADC_MANUAL_INTEN          // 使能 ADC 自动方式的中断
                        | C_ADC_MANUAL_FLAG;    // 清中断标志
*P_ADC_MIC_CTRL2 = C_ADC_CTRL_EN                // 使能 ADC 控制器
                        | C_ADC_MANUAL_MODE     // 选择自动采样模式
                        | C_ADC_AUTO_CLR        // 自动清除中断标志
                          ;
*P_ADC_INT_STATUS |= C_ADC_MANUAL_START;        // 启动 AD 转换
```

MIC 启动流程如图 4.30。步骤如下：

（1）　使能 MIC，选择为 MIC 通道；

（2）　设置 MIC 增益；

（3）　使能 MIC 中断；

（4）　使能 ADC 控制器。

注意这里只是 ADC 部分的设置，由于 MIC 输入的转换数据只能通过 DMA 读取，所以还要对 DMA 控制器进行设置，详见 DMA 相关章节。

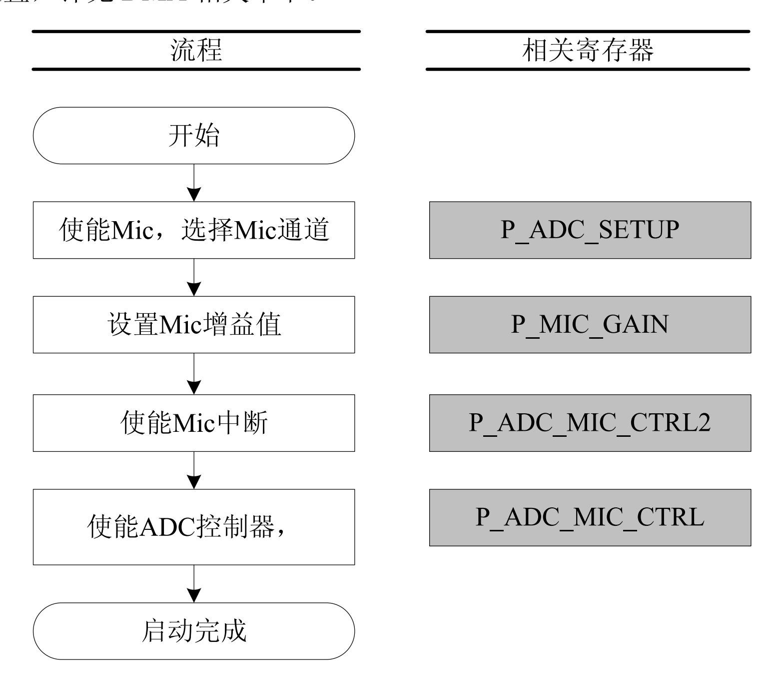


图 4.30 通用 A/D 的启动流程图

程序段参考如下：

```
*P_ADC_MIC_CTRL1 = C_ADC_MODE_MIC         // 使能 MIC
                   | C_ADC_MIC_SEL         // 选择 MIC 通道
                   | C_MIC_BIAS_EN         // MIC 偏压使能
                   | C_MIC_BOOST_EN        // MIC 前级放大器使能
                          ;
*P_ADC_MIC_GAIN = C_MIC_GAIN_330;          // 增益值设置为 33.0db
*P_ADC_INT_STATUS = C_ADC_MICOV_INTEN // 使能 MIC 溢出中断
                   | C_ADC_MICOV_FLAG;     // 清 MIC 溢出中断标志
*P_ADC_MIC_CTRL2 = C_ADC_CTRL_EN;          // 使能 ADC 控制器
```

3、 中断

SPCE3200 的 40 个中断源中，其中有两个关于 ADC 的中断源：通用 A/D 中断和 MIC 溢出/FIFO 溢出中断，这两个中断都通过 P_ADC_INT_STATUS 寄存器使能，进入中断后通过 P_ADC_INT_STATUS 寄存器清除中断标志。

注：当 ADC 作为通用 A/D 使用时，如果通过 P_ADC_MIC_CTRL2 寄存器的 bit23 把自动清除中断标志位使能，可不用专门的清除中断标志操作。

当 ADC 作为通用 A/D 使用时，完成一次 A/D 转换，引发一次 A/D 中断，这样，可以根据 P_ADC_INT_STATUS 的 bit31 或者 bit29 判断 A/D 转换是否完成。

中断方式中断服务程序段：

```
//=======================================================
// 语法格式：void IRQ58(void)
// 功能描述: 通用 ADC 中断服务函数
// 入口参数: 无
// 出口参数: 无
//=======================================================
void IRQ58(void)
{
    unsigned int Data = 0;
    float vDataConvert = 0.0000;
    *P_ADC_INT_STATUS |= C_ADC_AUTO_FLAG;   // 清中断标志
    Data = *P_ADC_AUTO_DATA;                // 取出转换数据
    Data = Data >> 20;                      // 把转换数据移到低 12 位
    vDataConvert = Data * 3.3 / 0xfff;      // 计算电压值
}
```

4.7.7 注意事项

使用 SPCE3200 的 ADC 时需要注意下面几点：

- 通用 A/D 输入通道输入的最大电压不能大于参考电压
- 参考电压不能超过 3.3V（I/O 口电压选择为 3.3V 时）

4.8 UART

4.8.1 概述

SPCE3200 具有一个标准的通用异步串行通讯模块 UART (Universal Asynchronous Receiver/Transmitter)，使其与 MCU 之间具有更强的通讯能力，而且通讯更加灵活。

SPCE3200 的 UART 数据帧格式如图 4.31，可以看出，在 UART 的数据传输时，先传输低位后传输高位；数据位位数可以设置；可以编程设置奇偶校验位。

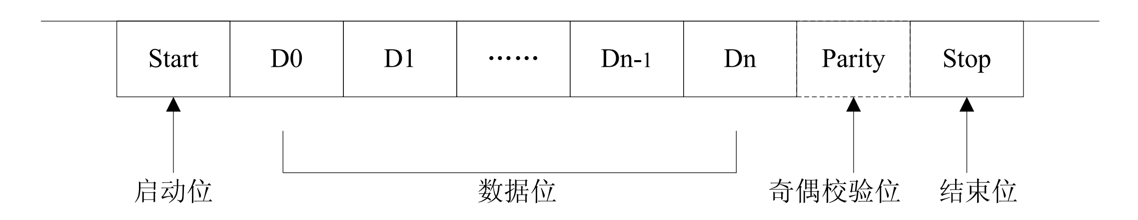


图 4.31 UART 数据帧格式

4.8.2 特性

SPCE3200 具有如下特性：

- 最大波特率可达到 460.8Kbps
- 可编程设置发送等待时间和接收等待（潜伏）时间
- 内嵌 2 字节发送 FIFO 和 8 字节接收 FIFO
- 发送、接收和接收超时中断可独立屏蔽或者使能
- 错误启动位检测
- 连接中止产生及检测
- 完全可编程串行接口：
    - 数据位可被设置为 5bit、6bit、7bit 或者 8bit
    - 奇、偶、无校验产生和检测
    - 1 位或者 2 位的停止位产生

4.8.3 引脚描述

UART 引脚的描述如表 4.75。

表 4.75 UART 引脚描述

引脚名称	引脚号	引脚属性	引脚功能
UART_TX	39	O	UART 发送脚
UART_RX	40	I	UART 接收脚

注意：在与 PC 机通讯时，需要外接电平转换电路。

4.8.4 结构框图

UART 的结构框图如图 4.32，UART 主要由 UART 时钟、中断控制器、FIFO（2 个字节的发送 FIFO 和 8 字节的接收 FIFO）、波特率发生器和 UART 发送/接收控制器构成。UART 时钟为 UART 模块的各个单元提供时钟；中断控制器控制使能发送或者接收中断；FIFO 为发送或者接收数据的缓存，当发送数据时，先把发送数据填入 FIFO，经移位寄存器一位一位移入 UART 控制器控制发送，当接收数据时，移位寄存器先把接收数据一位一位的填入 FIFO，再通过中断控制读取接收数据；波特率发生器提供发送或者接收数据的速率，当 SPCE3200 向其他 MCU 发送数据或者从其他 MCU 接收数据时，必须和其他 MCU 的波特率相同；UART 发送/接收控制器主要控制数据的发送和接收，包括奇偶校验位的设置、数据/停止位的设置等。

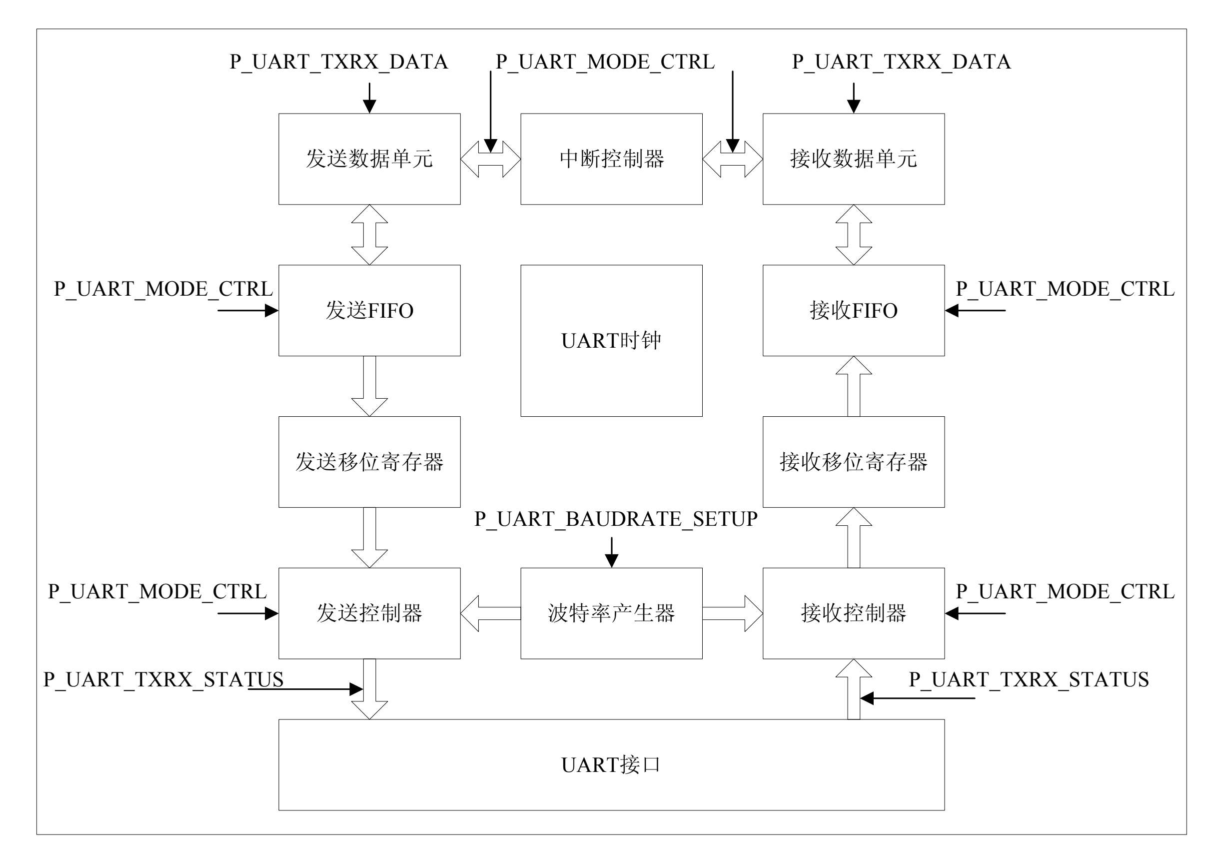


图 4.32 UART 结构框图

4.8.5 寄存器描述

UART 模块共有 12 个寄存器，如表 4.76 所示。

表 4.76 UART 相关寄存器列表

寄存器名称	助记符	地址
UART GPIO 设置寄存器	P_UART_GPIO_SETUP	0x88200040
UART GPIO 输入数据寄存器	P_UART_GPIO_INPUT	0x88200074
UART GPIO 外部中断寄存器	P_UART_GPIO_INT	0x88200094
UART 接口选择寄存器	P_UART_INTERFACE_SEL	0x88200000
UART 时钟配置寄存器	P_UART_CLK_CONF	0x8821005C
UART 控制寄存器	P_UART_MODE_CTRL	0x88150008
UART 波特率设置寄存器	P_UART_BAUDRATE_SETUP	0x8815000C
UART 收发状态寄存器	P_UART_TXRX_STATUS	0x88150010
UART 错误状态寄存器	P_UART_ERR_STATUS	0x88150004
UART 收发数据寄存器	P_UART_TXRX_DATA	0x88150000
UART 唤醒状态寄存器	P_UART_WAKEUP_STATUS	0x88210110



如果 UART 接口复用为 GPIO 功能，可以对表 4.76 的前三个寄存器操作。寄存器的各位对应引脚关系如表 4.142：

表 4.77 UART 接口复用为 GPIO 引脚与寄存器位对应关系

引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
UART_TX	160	bit[0]	可以
UART_RX	161	bit[1]	可以

UART GPIO 设置寄存器：P_UART_GPIO_SETUP(0x88200040)

UART GPIO 设置寄存器的功能是：当 UART 的端口作为通用输入输出口（GPIO）使用时设置输出使能、设置为上/下拉电阻输入、输出数据。

表 4.78 P_UART_GPIO_SETUP(0x88200040)

位	b31~b26	b25~b24	b23~b18	b17~b16	b15~b10	b9~b8	b7~b2	b1~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W
默认值	-	0x3	-	0	-	0	-	0
名称	-	XUART_PD	-	XUART_PU	-	UART_OE	-	UART_O

XUART_PD b25~b24 UART GPIO 下拉使能位：

0：不使能为下拉口

1：使能为下拉口

XUART_PU b17~b16 UART GPIO 上拉使能位：

0：使能为上拉口

1：不使能为上拉口

UART_OE b9~b8 UART GPIO 输出使能位：

0：不使能位输出口

1：使能为输出口

UART_O b1~b0 UART GPIO 输出数据

注意：b0、b8、b16、b24 控制的是 RX；b1、b9、b17、b25 控制的是 TX。

UART GPIO 输入数据寄存器：P_UART_GPIO_INPUT(0x88200074)

UART GPIO 输入数据寄存器保存 UART 接口复用为 GPIO 时的外部输入数据。

表 4.79 P_UART_GPIO_INPUT(0x88200074)

位	b31~b26	b17~b16	b1~b0
读/写	-	R	-

默认值	-	0	-
名称	-	UART_IN	-

UART_IN　　b17~b16　　UART 作为 GPIO 功能时的输入数据

UART GPIO 外部中断寄存器：P_UART_GPIO_INT(0x88200094)

当 UART 端口作为 GPIO 时，设置外部中断使能、中断触发沿设置和中断标志的清除。

表 4.80　P_UART_GPIO_INT(0x88200094)

位	b31~b26	b25~b24	b23~b18	b17~b16	b15~b10	b9~b8	b7~b2	b1~b0
读/写	-	R/W	-	R/W	-	W	-	W
默认值	-	0	-	0	-	0	-	0
名称	-	UART_FI	-	UART_RI	-	UART_FIEN	-	UART_RIEN

UART_FI　　b25~b24　　UART GPIO 下降沿中断标志位：

读 0：没有发生下降沿中断

读 1：发生下降沿中断

写 0：无意义

写 1：清除中断标志

UART_RI　　b17~b16　　UART GPIO 上升沿中断标志位：

读 0：没有发生上升沿中断

读 1：发生上升沿中断

写 0：无意义

写 1：清除中断标志

UART_FIEN　　b9~b8　　UART GPIO 下降沿中断使能位：

0：不使能下降沿中断

1：使能下降沿中断

UART_RIEN　　b1~b0　　UART GPIO 上升沿中断使能位：

0：不使能上升沿中断

1：使能上升沿中断

UART 时钟配置寄存器：P_UART_CLK_CONF(0x8821005C)

通过 UART 时钟配置寄存器可以停止/打开 UART 时钟或者复位/不复位 UART 模块，在使能 UART 前需先打开 UART 时钟。

表 4.81　P_UART_CLK_CONF(0x8821005C)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	1
名称	-	UART_RST	UART_STOP

UART_RST b1 UART 模块时钟复位位：

0：UART 模块时钟复位

1：UART 模块时钟不复位

UART_STOP b0 UART 模块时钟使能位：

0：UART 模块时钟停止

1：UART 模块时钟使能

UART 接口选择寄存器：P_UART_INTERFACE_SEL(0x88200000)

UART 接口选择寄存器的功能是：选择 UART 的端口作为普通输入输出口（GPIO）或者作为 UART 的专用发送/接收口。

表 4.82 P_UART_INTERFACE_SEL(0x88200000)

位	b31~b25	b24	b23~b0
读/写	-	R/W	-
默认值	-	1	-
名称	-	SW_UART	-

SW_UART b24 UART 的端口功能选择位：

0：作为 GPIO

1：作为 UART 的发送/接收口

UART 波特率设置寄存器：P_UART_BAUDRATE_SETUP(0x8815000C)

用来设置 UART 发送或者接收的波特率。

表 4.83 P_UART_BAUDRATE_SETUP(0x8815000C)

位	b31~b8	b7~b0
读/写	-	W
默认值	-	0xe9(115Kbps)
名称	-	BAUD_RATE

BAUD_RATE　　b[7:0]　　设置传输波特率。

波特率的计算公式如下：

波特率=Fosc/BAUD_RATE -1，其中 Fosc 为 27MHz，BAUD_RATE 为 b[7:0]设置的数据

注：由于 SPCE3200 的 UART 最高数据传输波特率为 460.8Kbps，所以 P_UART_BaudRate 最小可以设置为 Fosc/(460800+1)，最小可以设置为 27000000/(460800+1)=58（0x3A）。

UART 控制寄存器：P_UART_MODE_CTRL(0x88150008)

UART 控制寄存器用来控制 UART 的使能、UART 发送/接收中断的使能、传输数据位数的选择、停止位的选择、奇偶校验的使能及选择。

表 4.84　P_UART_MODE_CTRL(0x88150008)

位	b15	b14	b13	b12	b11~b8			
读/写	W	W	W	W	-			
默认值	0	0	0	1	-			
名称	RIEN	TIEN	RT	UEN	-			
位	b7	b6	b5	b4	b3	b2	b1	b0
读/写	-	W		W	W	W	W	W
默认值	-	3		1	0	1	1	0
名称	-	WLSEL		FEN	SBSEL	PSEL	PEN	SB

RIEN　　b15　　UART 接收中断使能位：

0：屏蔽 UART 接收中断

1：使能 UART 接收中断

TIEN　　b14　　UART 发送中断使能位：

0：屏蔽 UART 发送中断

1：使能 UART 发送中断

RT　　b13　　UART 接收超时中断使能：

0：屏蔽 UART 接收超时中断

1：使能 UART 接收超时中断

UEN　　b12　　UART 使能位：

0：禁止

1：使能

WLSEL　　b6~b5　　数据位长度定义，设置发送/接收一帧数据中数据的位数：

00：5bits



		01：6bits 10：7bits 11：8bits
FEN	b4	FIFO Buffer 使能位，在发送或者接收的过程中使能 FIFO Buffer，当该位被清零，FIFO 将成为 1 字节的保持寄存器： 0：禁止 1：使能
SBSEL	b3	停止位长度选择。当这一位被设置为 1 时，在发送一帧数据的结尾有 2bits 停止位。接收逻辑检测不到接收数据帧中的 2bit 停止位： 0：选择 1bit 停止位 1：选择 2bits 停止位
PSEL	b2	奇偶校验选择位： 0：选择奇校验 1：选择偶校验 当这一位被设置为 1 时，产生一个偶校验位并在发送或者接收过程中进行偶校验检测； 当这一位被设置为 0 时，产生一个寄校验位并在发送或者接收过程中进行寄校验检测。 该位只有当 P_UART_MODE_CTRL 的 b1（PEN）设置为 1 时设置有效。
PEN	b1	奇偶校验使能位： 0：无校验 1：使能校验
SB	b0	传输中止设置位： 0：正常传输 1：传输中止信号 当这一位被设置为 1 时，完成当前的数据传输之后，在发送输出脚发送一个持续的低电平。这个低电平至少要持续传输一帧数据所需要的时间，以完成中止过程。在中止的过程中 FIFO 中的内容不被影响。正常使用时，这一位必须被清 0。

UART 收发状态寄存器：P_UART_TXRX_STATUS(0x88150010)

UART 收发状态寄存器可以设置 UART 的中断、FIFO 等状态。

表 4.85　P_UART_TXRX_STATUS(0x88150010)

位	b15	b14	b13	b7	b6	b5	b4	b3

读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	RI	TI	RT	TE	RF	TF	RE	BY

RI　　b15　　UART 接收中断标志位：

　　0：没有发生 UART 接收中断

　　1：发生 UART 接收中断

TI　　b14　　UART 发送中断标志位：

　　0：没有发生 UART 发送中断

　　1：发生 UART 发送中断

RT　　b13　　UART 接收超时中断标志位：

　　0：没有发生 UART 接收超时中断

　　1：发生 UART 接收超时中断

TE　　b7　　发送 FIFO 空标志：

　　0：非空

　　1：空

该位依赖于 FIFO Buffer 是否使能：

如果 FIFO Buffer 没有使能，当发送保持寄存器为空时该位被置 1；

如果 FIFO Buffer 使能，当发送 FIFO 为空时该位被置 1。

RF　　b6　　接收 FIFO 满标志位：

　　0：非满

　　1：满

该位依赖于 FIFO Buffer 是否使能：

如果 FIFO Buffer 没有使能，当接收保持寄存器为满时该位被置 1；

如果 FIFO Buffer 使能，当接收 FIFO 为满时该位被置 1。

TF　　b5　　发送 FIFO 满标志位：

　　0：非满

　　1：满

如果 FIFO Buffer 没有使能，当发送保持寄存器为慢时该位被置 1；

如果 FIFO Buffer 使能，当发送 FIFO 为满时该位被置 1。

RE　　b4　　接收 FIFO 空标志位：

　　0：非空

　　1：空

如果 FIFO Buffer 没有使能，当接收保持寄存器为空时该位被置 1；



如果 FIFO Buffer 使能，当接收 FIFO 为空时该位被置 1。

BY　　b3　　忙标志位：

0：非忙

1：忙

如果 UART 正在忙于传输数据，该位被置 1，直到所有的数据从移位寄存器传输完（包括所有的停止位）。

UART 错误状态寄存器 P_UART_ERR_STATUS(0x88150004)

UART 错误状态寄存器用来存放 UART 通讯过程中的各种错误标志，包括溢出错误、连接断开错误、奇偶校验错误、帧传输错误等，给该寄存器的相应位写 1 可清除错误标志。

表 4.86　P_UART_ERR_STATUS(0x88150004)

位	b31~b16	b15	b14~b4	b3	b2	b1	b0
读/写	-	R	-	R/W	R/W	R/W	R/W
默认值	-	1	-	0	0	0	0
名称	-	RXD	-	OE	BE	PE	FE

RXD　　b15　　UART 接收信号位。在接收的过程中，为低电平，接收完成后，被置高电平。

OE　　b3　　UART 传输溢出标志位，当 UART 接收到数据且接收 FIFO 已经满时该标志被置 1：

读 0：没有发生 UART 传输溢出

读 1：发生 UART 传输溢出

写 0：无意义

写 1：清除溢出错误标志

BE　　b2　　UART 传输中止错误标志位，当 UART 接收到数据输入持续低电平一个完整字（起始位+数据位+奇偶校验位+结束位）传输时间，将会检测到中止并将该标志置 1：

读 0：没有发生传输中止错误

读 1：发生传输中止错误

写 0：无意义

写 1：清除中止错误标志

PE　　b1　　UART 奇偶校验错误标志位，当 UART 接收到数据特征中奇偶校验位与通过 P_UART_MODE_CTRL 设置不匹配，该标志置 1。每次通过数据寄存器读接收数据时，该位都会被刷新一次，所以在读出数据之后必须要检测该位：

读 0：没有发生奇偶校验错误

读 1：发生奇偶校验错误

写 0：无意义

写 1：清除奇偶校验错误标志

FE　　b0　　UART 的帧传输错误标志位，当 UART 接收到数据没有一个有效的结束位，该标志置 1。每次通过数据寄存器读接收数据时，该位都会被刷新一次，所以在读出数据之后必须要检测该位：

读 0：没有发生帧传输错误

读 1：发生帧传输错误

写 0：无意义

写 1：清除帧传输错误标志

UART 收发数据寄存器：P_UART_TXRX_DATA(0x88150000)

UART 收发数据寄存器用来保存 UART 准备发送的数据或者接收到的数据。

表 4.87　P_UART_TXRX_DATA(0x88150000)

位	b31~b8	b7~b0
读/写	-	R/W
默认值	-	0x00
名称	-	UART_DATA

UART_DATA　　b7~b0　　读：UART 接收到的数据

写：UART 准备发送的数据

UART 唤醒状态寄存器：P_UART_WAKEUP_STATUS(0x88210110)

UART 唤醒状态寄存器显示是否发生了 UART 唤醒或者 USB 唤醒。（该寄存器与 USB 唤醒状态寄存器是同一个地址）。

表 4.88　P_UART_WAKEUP_STATUS(0x88210110)

位	b31~b2	b1	b0
读/写	-	R	R
默认值	-	0	0
名称	-	UART_WAKEUP	USB_WAKEUP

UART_WAKEUP　　b1　　UART 唤醒状态位：

0：没有发生 UART 唤醒



1：发生了 UART 唤醒

USB_WAKEUP b0 USB 唤醒标志位：

0：没有发生 USB 唤醒

1：发生了 USB 唤醒

4.8.6 基本操作

如图 4.32 UART 的结构框图，UART 包括两个基本操作：发送和接收。

UART 的发送

UART 发送包括查询和中断两种方式。

UART 查询方式发送流程如图 4.33。发送前先要打开 UART 时钟；设置好发送波特率；使能 UART 及发送中断；只要 UART 发送不"忙"，就可以写发送数据，等待发送，发送完成后"忙"信号被清零。

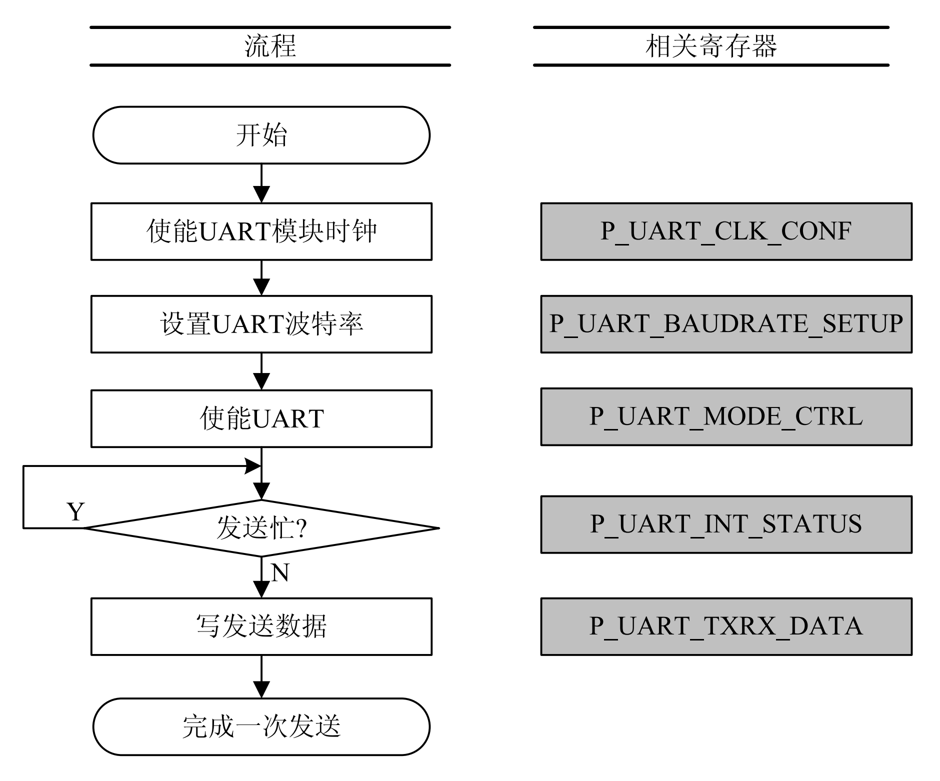


图 4.33 UART 查询方式发送流程图

如果要发送多次数据，发送完成后直接回到判断发送是否忙循环即可。

程序段如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
```

```
        unsigned int uiData = 1;                            // 将要发送的数据

        *P_UART_CLK_CONF = C_UART_CLK_EN
                         | C_UART_RST_DIS;                  // 使能 UART 模块时钟
        *P_UART_INTERFACE_SEL = C_UART_PORT_SEL;            // 选择端口作为 UART 使用，默认
        *P_UART_BAUDRATE_SETUP = 0xEA;                      // 115200
        *P_UART_MODE_CTRL = C_UART_NO_PARITY                // 无奇偶校验
                         | C_UART_STOP_1BIT                 // 1bit 停止位
                         | C_UART_DATA_8BIT                 // 8bit 数据位
                         | C_UART_CTRL_EN;                  // 使能 UART
        while(1)
        {
            if((*P_UART_TXRX_STATUS
                & (C_UART_BUSY_FLAG | C_UART_TXFIFO_FULL)) == 0)
            {
                *P_UART_TXRX_DATA = uiData;                 // 写发送数据
                uiData++;
            }
        }
}
```

UART 中断方式发送包括两部分：主程序部分和中断服务程序部分。

主程序流程如图 4.34，主要进行初始化操作，包括 UART 模块时钟的使能，波特率的设置和使能 UART 及 UART 发送中断。

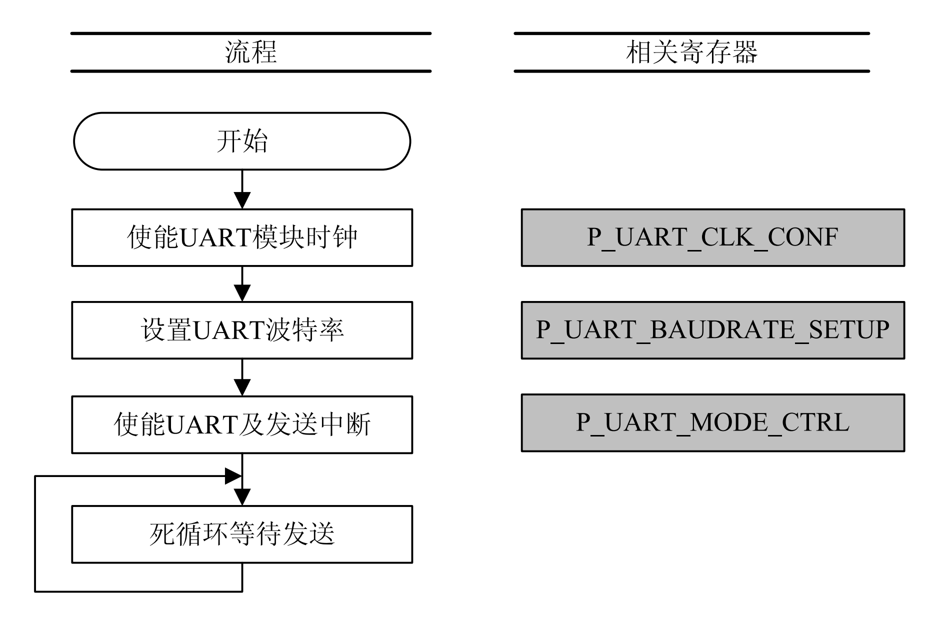




图 4.34 UART 中断方式发送主程序流程图

程序段如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
    *P_INT_MASK_CTRL1 = ~C_INT_UART_DIS;        // 打开 UART 中断
    *P_UART_CLK_CONF = C_UART_CLK_EN            // 使能 UART 模块时钟
                      | C_UART_RST_DIS;
    *P_UART_BAUDRATE_SETUP = 0xEA;              // 选择波特率 115200bps
    *P_UART_MODE_CTRL = C_UART_NO_PARITY        // 无奇偶校验
                      | C_UART_STOP_1BIT        // 1bit 停止位
                      | C_UART_DATA_8BIT        // 8bit 数据位
                      | C_UART_TX_INTEN         // 发送中断使能
                      | C_UART_CTRL_EN;         // UART 使能
    while(1);                                   // 等待中断
}
```

中断服务程序流程如图 4.35：UART 的中断服务程序不需要手动清除中断标志，而是在处理完中断服务程序时会自动清除中断标志。

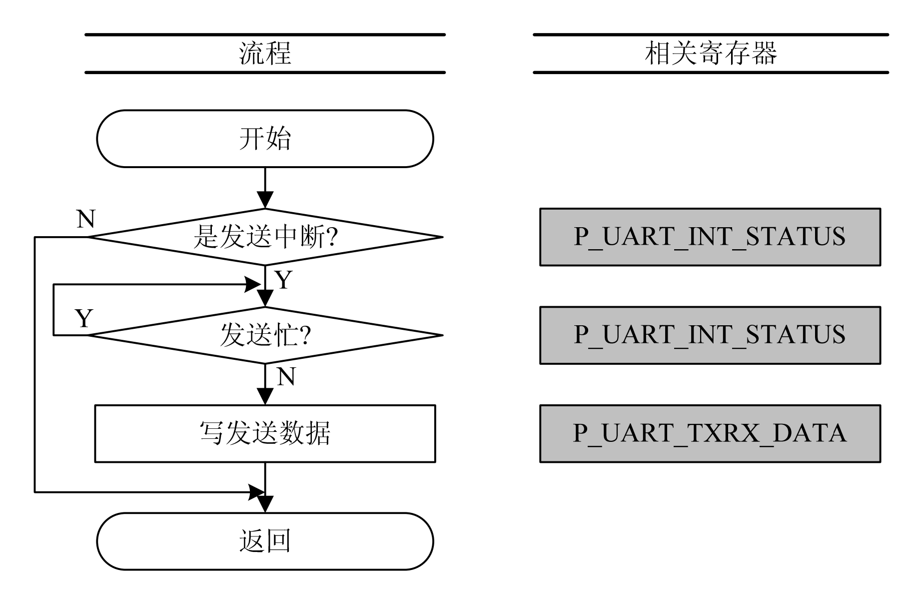


图 4.35 UART 中断方式发送中断服务程序流程图

程序段如下：

```
//====================================================
```

```
// 语法格式：void IRQ42(void)
// 功能描述: UART 中断服务函数
// 入口参数: 无
// 出口参数: 无
//=====================================================
void IRQ42(void)
{
    if(*P_UART_TXRX_STATUS & C_UART_TX_FLAG)
    {
        if((*P_UART_TXRX_STATUS
            & (C_UART_BUSY_FLAG | C_UART_TXFIFO_FULL)) == 0)
        {
            *P_UART_TXRX_DATA = 0xAA;            // 写发送数据
        }
    }
}
```

UART 的接收

UART 接收也包括查询和中断两种方式，分别介绍。

UART 查询方式发送流程如图 4.36。接收前先打开 UART 时钟；设置好接收波特率；使能 UART 及接收中断；如果接收中断来，通过 P_UART_TXRX_DATA 单元读出数据。

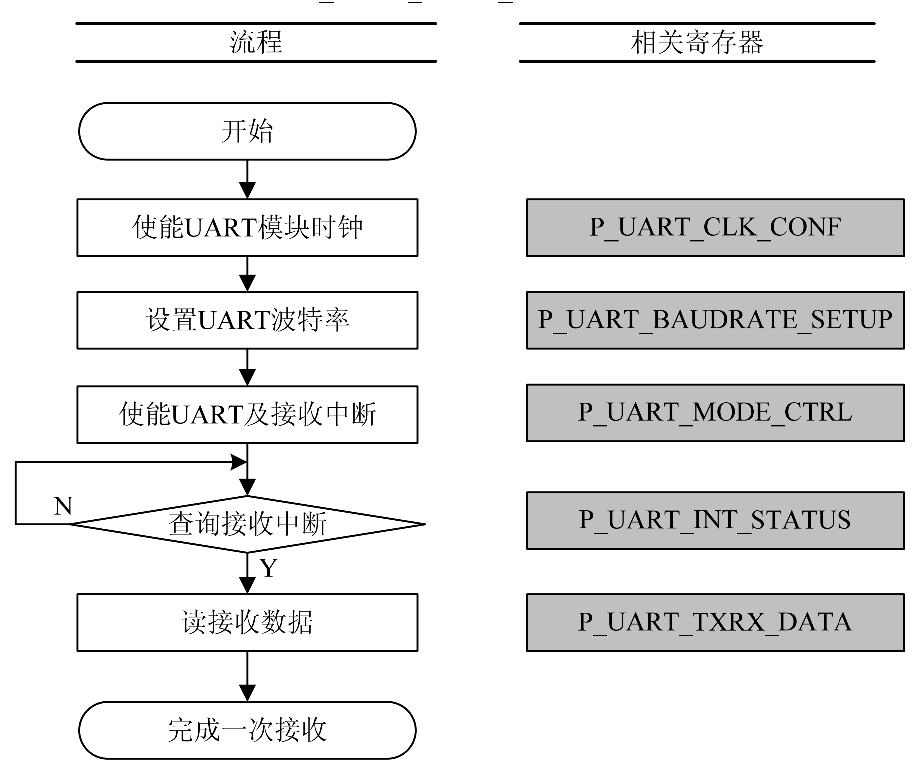


图 4.36 UART 接收流程图

如果要接收多次数据，接收完成读出数据后直接回到判断接收 FIFO 是否满循环即可。

程序段如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
int main(void)
{
    unsigned int uiData[10];                            // 定义数组接收数据
    unsigned int i = 0;

    *P_UART_CLK_CONF = C_UART_CLK_EN
                     | C_UART_RST_DIS;                  // 使能 UART 模块时钟
    *P_UART_INTERFACE_SEL = C_UART_PORT_SEL;            // 选择端口作为 UART 收发端口
    *P_UART_BAUDRATE_SETUP = 0xEA;                      // 设置波特率为 115200bps
    *P_UART_MODE_CTRL = C_UART_NO_PARITY                // 无奇偶校验
                    | C_UART_STOP_1BIT                  // 1bit 停止位
                    | C_UART_DATA_8BIT                  // 8bit 数据位
                    | C_UART_CTRL_EN                    // 使能 UART
                    | C_UART_RX_INTEN                   // 使能 UART 接收中断
                        ;
    while(1)
    {
        if(*P_UART_TXRX_STATUS & C_UART_RX_FLAG)        // 有数据？
        {
            uiData[i] = *P_UART_TXRX_DATA;              // 读接收数据
            i++;
            if(i == 9)
                i = 0;
        }
    }
}
```

UART 中断方式接收同样包括两部分：主程序部分和中断服务程序部分。

主程序流程如图 4.34，主要进行初始化包括 UART 模块时钟的使能，波特率的设置和打开 UART 及 UART 接收中断。

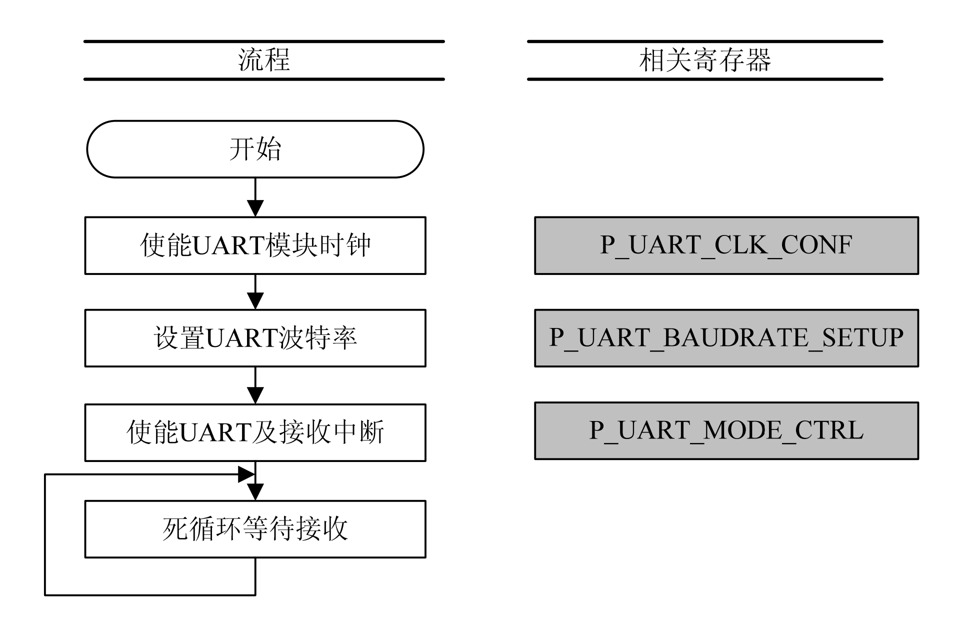


图 4.37　UART 中断方式接收主程序流程图

程序段如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
    *P_INT_MASK_CTRL1 = ~C_INT_UART_DIS;              // 打开 UART 中断
    *P_UART_CLK_CONF = C_UART_CLK_EN
                      | C_UART_RST_DIS;                // 使能 UART 模块时钟
    *P_UART_INTERFACE_SEL = C_UART_PORT_SEL;          // 选择端口作为 UART 端口
    *P_UART_BAUDRATE_SETUP = 0xEA;                    // 设置波特率为 115200bps
    *P_UART_MODE_CTRL = C_UART_NO_PARITY              // 无奇偶校验
                     | C_UART_STOP_1BIT               // 1bit 停止位
                     | C_UART_DATA_8BIT               // 8bit 数据位
                     | C_UART_CTRL_EN                 // UART 使能
                     | C_UART_RX_INTEN                // UART 接收中断使能
                       ;
    while(1);                                         // 等待接受中断
}
```

中断服务程序流程如图 4.35：同样，UART 的接收中断服务程序也不需要手动清除中断标志，而是在处理完中断服务程序时会自动清除中断标志。

程序段如下：

```
//=======================================================
// 语法格式：void IRQ42(void)
// 功能描述: UART 中断服务函数
// 入口参数: 无
// 出口参数: 无
//=======================================================
void IRQ42(void)
{
	unsigned int uiData;

	if(*P_UART_TXRX_STATUS & C_UART_RX_FLAG)	// 是否是接收中断？
	{
		uiData = *P_UART_TXRX_DATA;			// 读接收数据
	}
}
```

4.8.7 注意事项

使用 SPCE3200 的 UART 通讯时需要注意下面几点：

1、 通讯双方的波特率必须设置一致；

2、 通讯双方的奇偶校验方式要设置一致；

3、 通讯双方接收/发送的数据位要设置一致，即如果 SPCE3200 的数据位设置为 6bits，通讯另一方的数据位必须也能设置为 6bits 且必须设置为 6bits；

4、 计算波特率时需要注意，波特率=27MHz/BAUD_RATE -1，即 BAUD_RATE=Fosc/(波特率+1)，其中的 BAUD_RATE 为 P_UART_BAUDRATE_SETUP 寄存器设置的数据。

4.9 SPI

4.9.1 概述

SPCE3200 内嵌一个 SPI（Serial Peripheral Interface）接口控制器，该接口是一个同步、全双工串行接口，可以便利地与其他外设或者元器件通讯。

4.9.2 特性

SPCE3200 的 SPI 特性如下：

- 支持主/从模式的单字节和连续多字节数据传输
- 支持数据接收溢出错误检测
- 支持发送/接收中断请求

- 可编程主模式时钟的相位和极性
- 可选择数据采样时间
- 可编程主模式的时钟频率，该时钟频率可设置为 SPI 系统时钟（27MHz）的 1/2、1/4、1/8、1/16、1/32、1/64、1/128
- 内嵌 8Byte 的接收 FIFO 和发送 FIFO

4.9.3 引脚描述

SPCE3200 有四个 SPI 相关引脚，其中 SPI_CLK、SPI_TX、SPI_RX 为复用引脚，如表 4.89。

表 4.89 SPI 引脚列表

引脚名称	引脚号	引脚属性	引脚功能
NF_D4	111	I	SPI_RX 复用引脚，SPI 接收
NF_D5	110	O	SPI_TX 复用引脚，SPI 发送
NF_ALE	127	I/O	SPI_CLK 复用引脚，SPI 时钟信号
SPI_CSN	128	I/O	SPI 片选信号

SPCE3200 通过 SPI 与其他 SPI 设备或者器件通讯时，只需要把 SPCE3200 的 SPI_RX（NF_D4）与其它设备的发送脚（TX）连接，SPI_TX（NF_D5）与其他设备的接收脚（RX）连接，其他两个管脚分别对应连接即可。

4.9.4 结构框图

SPCE3200 的 SPI 模块结构框图如图 4.38，SPI 模块包括时钟产生电路，发送/接收的串/并转换电路及 FIFO、中断控制器和 SPI 控制器。时钟产生电路包括时钟分频器、极性/相位产生器，系统时钟通过分频器进行分频后，通过极性、相位产生器产生一个一定相位、一定极性和频率的时钟提供给 SPI 模块；并行发送数据的并/串转换电路和 FIFO 转换为串行数据后通过控制器控制发送，接收到的串行数据经过接收 FIFO 和串/并转换电路转换成并行数据，以便读取；中断控制器使能发送/接收中断，当有数据发送或者接收到数据时，向 CPU 发出中断请求；SPI 控制器控制选择为主模式或者从模式，并控制数据的发送和接收。

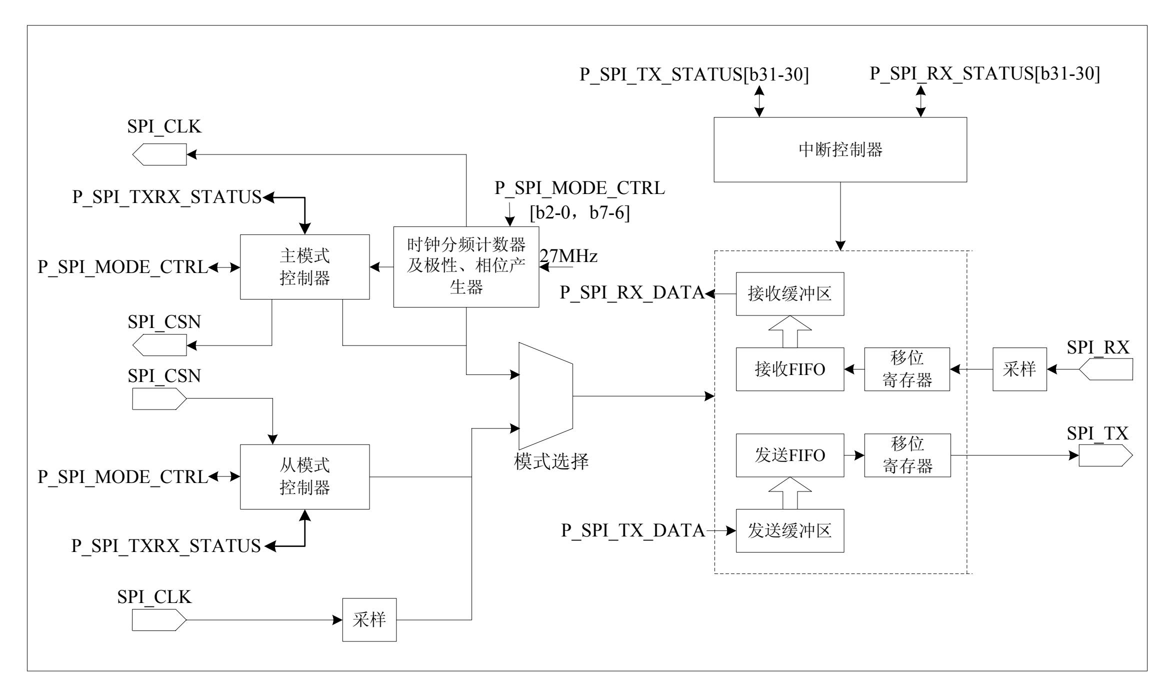


图 4.38 SPI 结构框图

4.9.5 SPI 描述

1、SPI 总线配置

SPI 是一个全双工的同步串行通讯接口，一个 SPI 总线可以连接多个主机和多个从机，但是在同一时刻只允许有一个主机操作总线。在一次数据传输中，主机向从机发送一字节的数据，同时，从机也向主机发送一字节数据。如图 4.39 为以 SPCE3200 为主机的 SPI 总线配置示例，SPI 总线的时钟是由主机 SPCE3200 产生的。

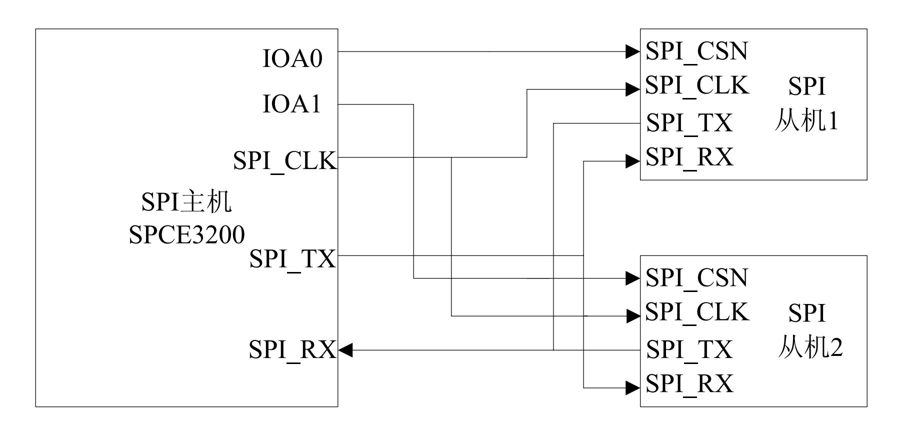


图 4.39 SPI 总线配置

2、SPI 数据传输

SPI 的数据传输根据时钟相位（SPH）和极性（SPO）的设置不同而不同，有下面四种情况：

（1）SPO=0，SPH=0

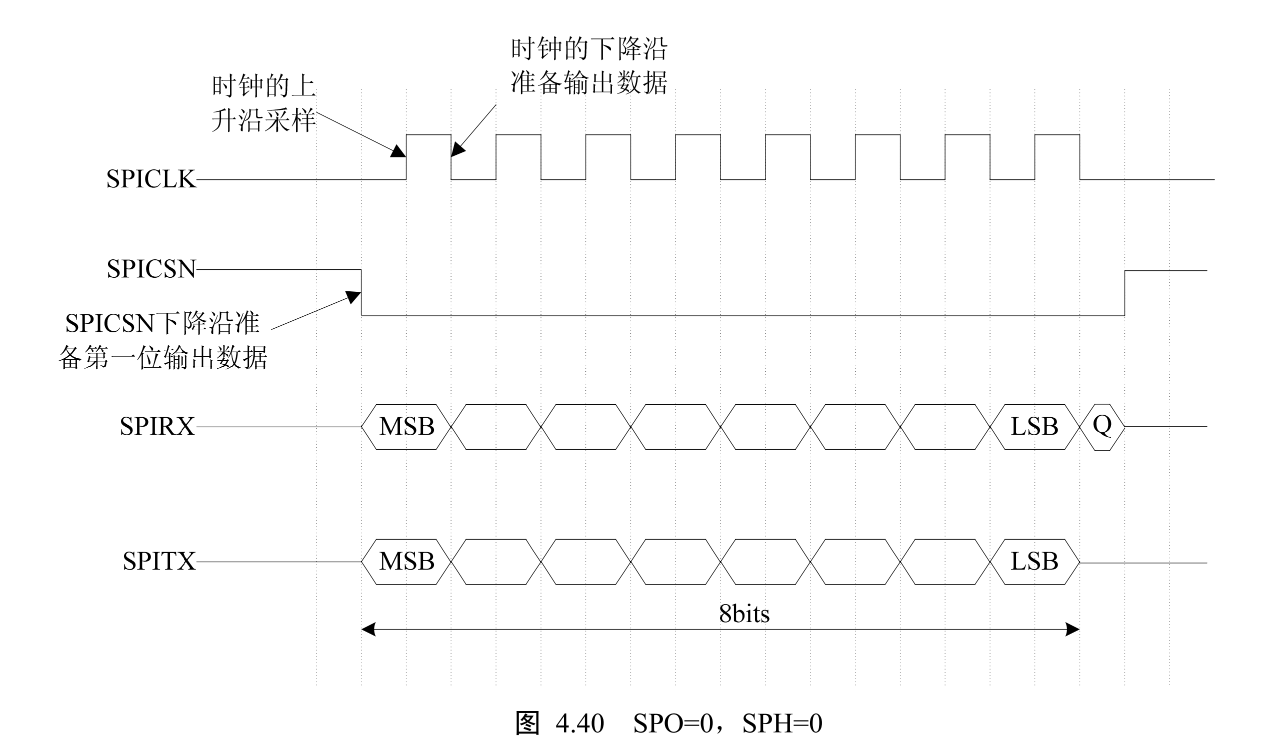


图 4.40 SPO=0，SPH=0

如图 4.40，当 SPO=0 时，时钟高电平有效，第一个时钟沿为上升沿；SPH=0，第一位数据在时钟的第一个时钟沿被采样，所以在时钟的第一个上升沿到来前必须把第一位传输数据输出到数据线上，其他位数据在时钟的下降沿被输出到数据线上，在时钟的上升沿被采样。

（2）SPO=0，SPH=1

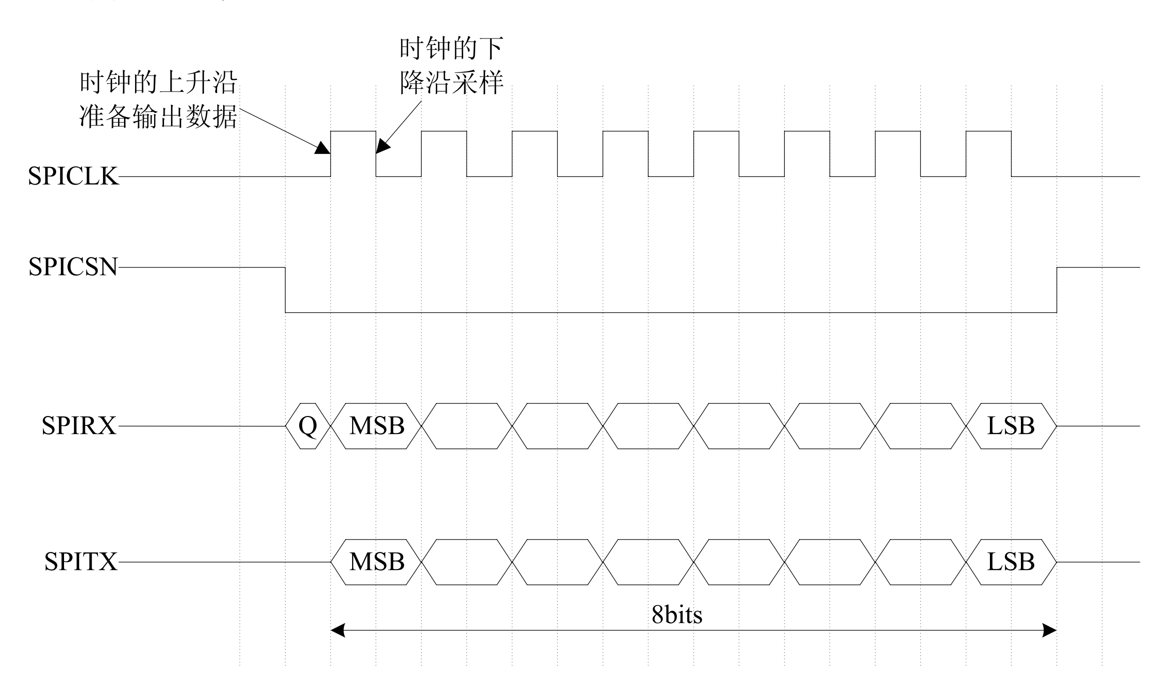


图 4.41 SPO=0，SPH=1

如图 4.41，当 SPO=0 时，时钟高电平有效，第一个时钟沿为上升沿；SPH=1，第一位数据在时钟的第二个时钟沿被采样，所以在数据传输过程中，所有位数据在时钟的上升沿输出到数据线上，在时钟的下降沿被采样。

（3）SPO=1，SPH=0

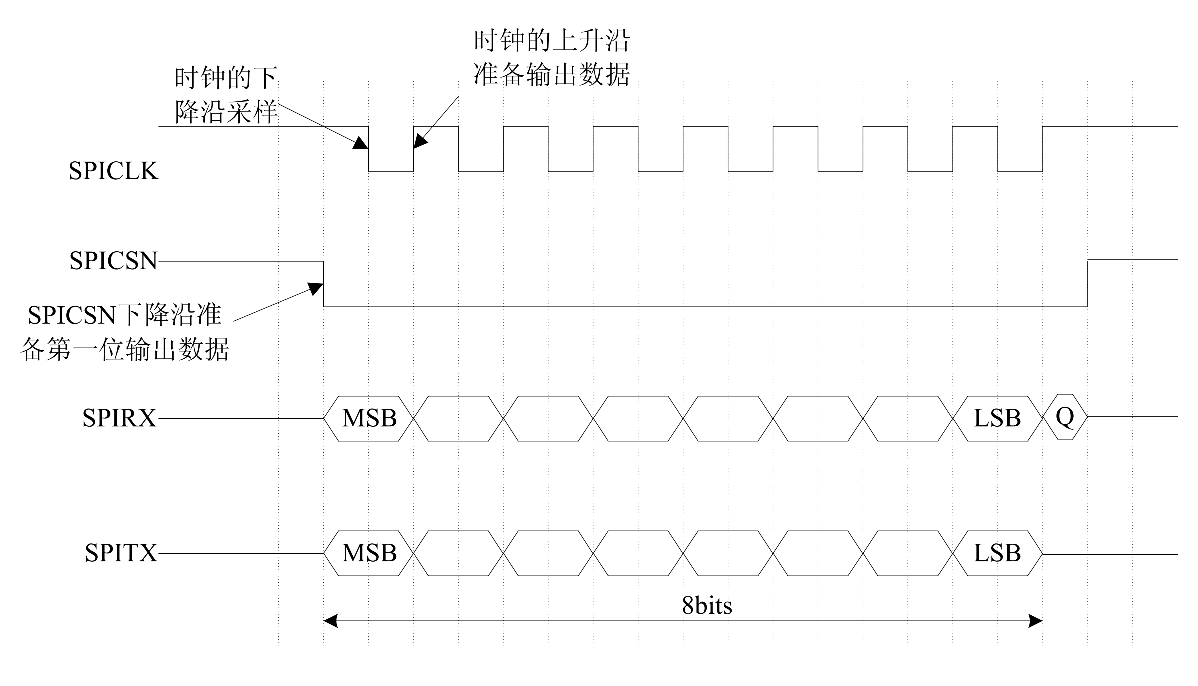


图 4.42 SPO=1，SPH=0

如图 4.42，当 SPO=1 时，时钟低电平有效，第一个时钟沿为下降沿；SPH=0，第一位数据在时钟的第一个时钟沿被采样，所以在数据传输过程中，第一位数据在第一个下降沿到来之前输出到数据线上，其他位数据在时钟的上升沿输出到数据线上，在时钟的下降沿被采样。

（4）SPO=1，SPH=1

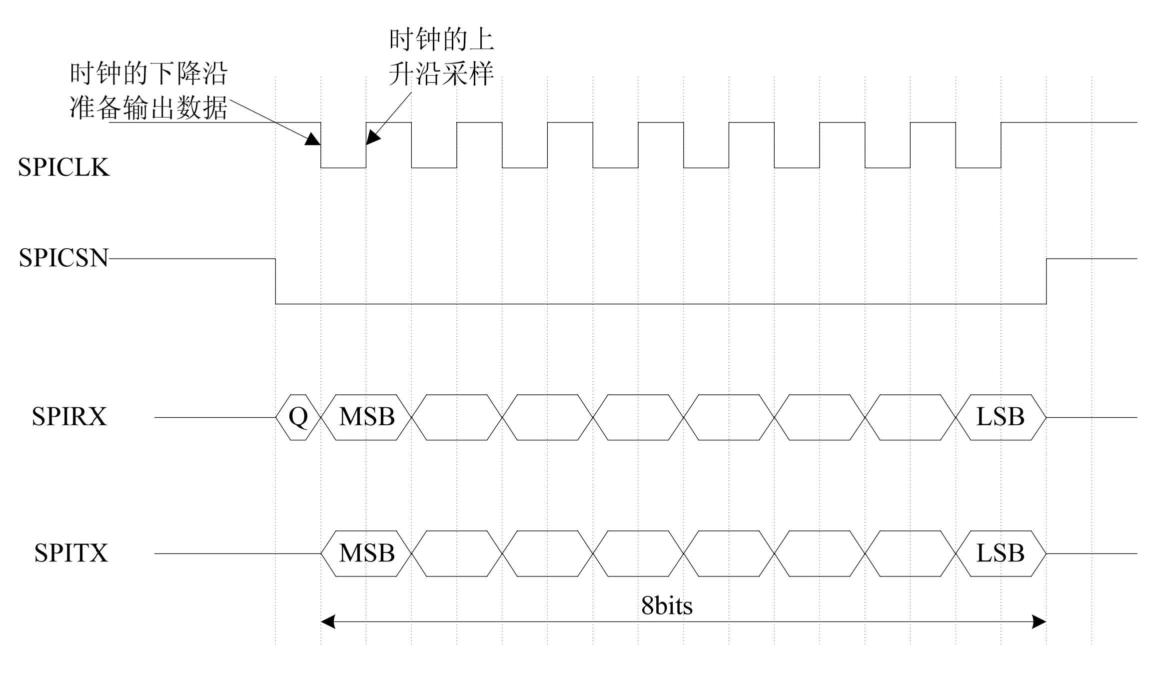


图 4.43 SPO=1，SPH=1

如图 4.43，当 SPO=1 时，时钟低电平有效，第一个时钟沿为下降沿；SPH=1，第一位数据在时钟的第二个时钟沿被采样，所有位数据在时钟的下降沿输出到数据线上，在时钟的上升沿被采样。

3、SPI 的应用

SPI 可应用于：

- 串行存储器，如 93C46 存储器；
- 串行外设，如 ADC、DAC、LCD 控制器、CAN 控制器和传感器等；
- 外部协处理器等。

4.9.6 寄存器描述

SPI 模块共有 8 个寄存器，如表 4.90。

表 4.90 SPI 相关寄存器列表

寄存器名称	助记符	地址
SPI 时钟配置寄存器	P_SPI_CLK_CONF	0x88210098
SPI 接口选择寄存器	P_SPI_INTERFACE_SEL	0x882000A4
SPI 控制寄存器	P_SPI_MODE_CTRL	0x88110000
SPI 发送状态寄存器	P_SPI_TX_STATUS	0x88110004
SPI 发送数据寄存器	P_SPI_TX_DATA	0x88110008
SPI 接收状态寄存器	P_SPI_RX_STATUS	0x8811000C
SPI 接收数据寄存器	P_SPI_RX_DATA	0x88110010
SPI 收发状态寄存器	P_SPI_TXRX_STATUS	0x88110014

SPI 时钟配置寄存器：P_SPI_CLK_CONF(0x88210098)

通过 SPI 时钟配置寄存器可以停止/打开 SPI 时钟或者复位/不复位 SPI 模块，在使能 SPI 前需先打开 SPI 模块时钟。

表 4.91 P_SPI_CLK_CONF(0x88210098)

位	b31~b2	b1	b0
读/写	-	W	W
默认值	-	1	0
名称	-	SPI_RST	SPI_STOP

SPI_RST　b1　SPI 模块时钟复位位：

0：SPI 模块时钟复位

1：SPI 模块时钟不复位



SPI_STOP　b0　　SPI 模块时钟使能位。

0：SPI 模块时钟停止

1：SPI 模块时钟使能

SPI 接口选择寄存器：P_SPI_INTERFACE_SEL(0x882000A4)

由于 SPI 的 SPI_Tx、SPI_Rx 和 SPI_CLK 与 Nand Flash 的接口复用，所以在使用这些端口前先要通过 SPI 接口选择存器设置这些复用接口为 SPI 功能。

表 4.92　P_SPI_INTERFACE_SEL(0x882000A4)

位	b31~b9	b8	b7~b0
读/写	-	R/W	-
默认值	-	0	-
名称	-	SPI_EN	-

SPI_EN　b8　　SPI 接口使能位：

0：复用接口不为 SPI 接口

1：复用接口作为 SPI 接口

SPI 控制寄存器：P_SPI_MODE_CTRL(0x88110000)

通过 P_SPI_MODE_CTRL 可以设置 SPI 的时钟频率、时钟的极性和相位、SPI 的主/从模式，进行 SPI 的使能等。

表 4.93　P_SPI_MODE_CTRL(0x88110000)

位	b31	b27	b26	b25	b7	b6	b2~b0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0	0
名称	SPI_EN	SPI_LB	SOFT_REST	SPI_MODE	SPI_CK_PH	SPI_CK_PO	SPI_CLOCK

SPI_EN　b31　　SPI 使能位：

0：禁止 SPI

1：使能 SPI

SPI_LB　b27　　SPI 循环送回模式选择位：

0：正常模式

1：SPIRX=SPITX

SOFT_REST　b26　　SPI 软复位使能位：

0：不进行软复位

1：软复位

SPI_MODE　　b25　　SPI 主/从模式选择位：

0：主模式

1：从模式

SPI_CK_PH　　b7　　SPI 时钟相位选择位（SPH）：

0：数据在第一个时钟沿被采样

1：数据在第二个时钟沿被采样

详细可参考图 4.40~图 4.43。

SPI_CK_PO　　b6　　SPI 时钟极性选择位（SPO）：

0：时钟高电平有效

1：时钟低电平有效

详细可参考图 4.40~图 4.43。

SPI_CLOCK　　b[2:0]　　SPI 主模式的时钟选择位：

000：PCLK/2

001：PCLK/4

010：PCLK/8

011：PCLK/16

100：PCLK/32

101：PCLK/64

110：PCLK/128

注：PCLK 是 APB 总线的主频，为 27MHz。

SPI 收发状态寄存器：P_SPI_TXRX_STATUS(0x88110014)

通过 P_SPI_TXRX_STATUS 可以设置 SPI 的清除中断标志方式、数据溢出模式，并可以得到 SPI 总线是否忙、发送/接收 FIFO 的状态信息等。

表 4.94　P_SPI_TXRX_STATUS(0x88110014)

位	b31	b30	b4	b3	b2	b1	b0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0	0
名称	SPI_OW_MODE	SPI_SMART	SPI_BUSY	RX_FULL	RX_EMPTY	TX_FULL	TX_EMPTY

SPI_OW_MODE　　b31　　SPI 溢出模式设置位：

0：当发生溢出时跳过数据

1：当发生溢出时重写数据

SPI_SMART　　b30　　SPI 清中断标志方式选择位：

0：手动清中断标志
1：自动清中断标志

SPI_BUSY b4 SPI 总线状态位：
0：空闲
1：忙

RX_FULL b3 SPI 接收 FIFO 满标志位：
0：非满
1：满

RX_EMPTY b2 SPI 接收 FIFO 空标志位：
0：空
1：非空

TX_FULL b1 SPI 发送 FIFO 满标志位：
0：满
1：非满

TX_EMPTY b0 SPI 发送 FIFO 空标志位：
0：非空
1：空

SPI 发送状态寄存器：P_SPI_TX_STATUS(0x88110004)

通过 SPI 发送状态寄存器可以设置 SPI 的发送中断等。

表 4.95 P_SPI_TX_STATUS(0x88110004)

位	b31	b30	b29~b28	b27	b26~b0
读/写	R/W	W	-	R	-
默认值	0	0	-	0	-
名称	TX_FLAG	TX_EN	-	TX_EMPTY_FLAG	-

TX_FLAG b31 SPI 发送中断标志位：
读 0：没有发生 SPI 发送中断
读 1：发生 SPI 发送中断
写 0：无意义
写 1：清除中断标志

TX_EN b30 SPI 发送中断使能位：
0：屏蔽 SPI 发送中断
1：使能 SPI 发送中断

TX_EMPTY_FLAG　b27　SPI 发送 FIFO 空标志位：
　0：非空
　1：空

SPI 接收状态寄存器：P_SPI_RX_STATUS(0x8811000C)

SPI 接收状态寄存器可以设置 SPI 的接收中断等。

表 4.96　P_SPI_RX_STATUS(0x8811000C)

位	b31	b30	b29~b28	b27	b26	b25~b0
读/写	R/W	W	-	R	R/W	-
默认值	0	0	-	0	0	-
名称	RX_FLAG	RX_EN	-	RX_FULL_FLAG	RX_OR_ERR	-

RX_FLAG　b31　SPI 接收中断标志位：
　读 0：没有发生 SPI 接收中断
　读 1：发生了 SPI 接收中断
　写 0：无意义
　写 1：清除该中断标志

RX_EN　b30　SPI 接收中断使能位：
　0：屏蔽 SPI 接收中断
　1：使能 SPI 接收中断

RX_FULL_FLAG　b27　SPI 接收 FIFO 满标志位：
　0：非满
　1：满

RX_OR_ERR　b26　SPI 接收溢出错误标志位：
　读 0：没有发生 SPI 接收溢出
　读 1：发生了 SPI 接收溢出
　写 0：无意义
　写 1：清除该标志

SPI 发送数据寄存器：P_SPI_TX_DATA(0x88110008)

SPI 发送数据寄存器用来保存 SPI 准备发送的数据。

表 4.97　P_SPI_TX_DATA(0x88110008)

位	b31~b8	b7~b0
读/写	-	W



默认值	-	0x00
名称	-	SPI_TX_DATA

SPI_TX_DATA　b7~b0　　SPI 准备发送的数据

SPI 接收数据寄存器：P_SPI_RX_DATA(0x88110010)

SPI 接收数据寄存器用来保存 SPI 接收到的数据。

表 4.98　P_SPI_RX_DATA(0x88110010)

位	b31~b8	b7~b0
读/写	-	R
默认值	-	0x00
名称	-	SPI_RX_DATA

SPI_RX_DATA　b7~b0　　SPI 接收到的数据

4.9.7　基本操作

SPCE3200 的 SPI 是一个同步、全双工串行接口，可以通过 P_SPI_MODE_CTRL 寄存器的 bit25 位设置为主机模式或从机模式，作为主机时输出的时钟速率、极性、相位可以通过 P_SPI_MODE_CTRL 的 bit0~bit2、bit6 和 bit7 编程设置。

不管 SPCE3200 工作在主机模式还是从机模式，分别有两种操作方式：查询方式和中断方式。

（1）查询方式，以主机为例

SPCE3200 作为主机查询方式传输数据的操作流程如图 4.44，操作可分为 SPI 初始化、发送和接收三个步骤。其中初始化包括使能 SPI 模块时钟；由于 SPI 的接口与 Nand Flash 及 SD 接口复用，所以需要设置为 SPI 接口；使能发送及接收中断；使能 SPI 准备发送或者接收数据；如图 4.44 。如果 SPI 总线不忙，且发送 FIFO 非满，填充发送数据准备发送；等待发送完成。如果接收 FIFO 非空，读取接收数据。

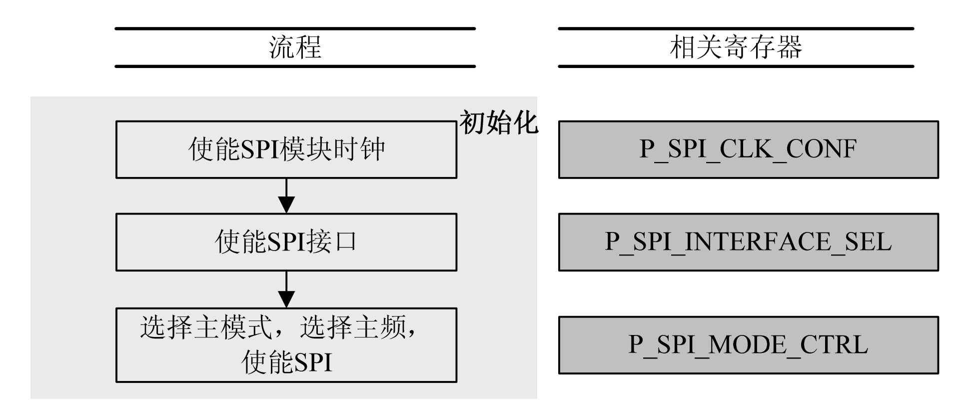


图 4.44 SPI 主机查询方式初始化

自发自收，即将 SPI_RX 与 SPI_TX 管脚相连，使用查询接收中断方式，参考程序：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
	unsigned int uiT_data[10] = {1,2,3,4,5,6,7,8,9,10};
	unsigned int uiR_data[10] = {0};
	unsigned int i = 0;

	*P_SPI_CLK_CONF = C_SPI_CLK_EN
	                 | C_SPI_RST_DIS;                    // 使能 SPI 模块时钟
	*P_SPI_INTERFACE_SEL = C_SPI_PORT_SEL;               // SPI 端口选择
	*P_SPI_MODE_CTRL = C_SPI_CLK_27MDIV128               // 选择系统时钟
	                | C_SPI_MASTER_MODE                  // 主模式
	                | C_SPI_NORMAL_MODE                  // 正常传输模式，相对 test 模式
	                | C_SPI_CTRL_EN;                     // SPI 使能
	*P_SPI_RX_STATUS = C_SPI_RX_INTEN                    // 使能 SPI 接受中断
	                | C_SPI_RX_FLAG;
	while(1)
	{
		*P_SPI_TX_DATA = uiT_data[i];                    // 发送
		while(!(*P_SPI_RX_STATUS & C_SPI_RX_FLAG));// 是否有接收中断
		*P_SPI_RX_STATUS |= C_SPI_RX_FLAG;               // 清除接收中断
		uiR_data[i] = *P_SPI_RX_DATA;                    // 接收数据
		i++;
		if(i == 10)
			i = 0;
	}
}
```

使用查询 FIFO 方式，参考程序：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
```

```
int main(void)
{
	unsigned int uiT_data[10] = {1,2,3,4,5,6,7,8,9,10};
	unsigned int uiR_data[10] = {0};
	unsigned int i = 0;
	unsigned int j = 0;

	*P_SPI_CLK_CONF = C_SPI_CLK_EN
	                  | C_SPI_RST_DIS;                   //  使能 SPI 模块时钟
	*P_SPI_INTERFACE_SEL = C_SPI_PORT_SEL;               // SPI 端口选择
	*P_SPI_MODE_CTRL = C_SPI_CLK_27MDIV128               //  选择系统时钟
	                 | C_SPI_MASTER_MODE                 //  主模式
	                 | C_SPI_NORMAL_MODE                 //  正常传输模式，相对 test 模式
	                 | C_SPI_CTRL_EN;                    // SPI 使能
	while(1)
	{
		*P_SPI_TXRX_STATUS = C_SPI_MANNUAL_CLR;
		while(*P_SPI_TXRX_STATUS & C_SPI_TXFIFO_NOTFULL)
		{
			*P_SPI_TX_DATA = uiT_data[i];            //  发送
			i++;
			if(i == 10)
				i = 0;
		}
		while(*P_SPI_TXRX_STATUS & C_SPI_RXFIFO_NOTEMPTY)
		{
			uiR_data[j] = *P_SPI_RX_DATA;
			j++;
			if(j == 10)
				j = 0;
		}
	}
}
```

中断方式和查询方式类似，在主程序里主要进行 SPI 的初始化操作，在中断服务程序里填写发

送数据或者读取接收数据。

例：自发自收，即将 SPI_RX 与 SPI_TX 管教相连，使用中断方式，参考程序：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"

int main(void)
{
    unsigned int uiT_data[10] = {1,2,3,4,5,6,7,8,9,10};
    unsigned int i = 0;

    *P_INT_MASK_CTRL1 = ~C_INT_SPI_DIS;              // 使能 SPI 中断
    *P_SPI_CLK_CONF = C_SPI_CLK_EN
                   | C_SPI_RST_DIS;                  // 使能 SPI 模块时钟
    *P_SPI_INTERFACE_SEL = C_SPI_PORT_SEL;           // SPI 端口选择
    *P_SPI_MODE_CTRL = C_SPI_CLK_27MDIV128           // 选择系统时钟
                   | C_SPI_MASTER_MODE               // 主模式
                   | C_SPI_NORMAL_MODE               // 正常传输模式，相对 test 模式
                   | C_SPI_CTRL_EN;                  // SPI 使能
    *P_SPI_RX_STATUS = C_SPI_RX_INTEN                // 使能 SPI 接受中断
                   | C_SPI_RX_FLAG;
    while(1)
    {
        while(*P_SPI_TXRX_STATUS & C_SPI_BUSY_FLAG);
        *P_SPI_TX_DATA = uiT_data[i];                // 发送
        i++;
        if(i == 10)
            i = 0;
    }
}
//=====================================================
// 语法格式：void IRQ43(void)
// 功能描述: SPI 中断服务函数
// 入口参数: 无
// 出口参数: 无
//=====================================================
void IRQ43(void)
```

```
{
    unsigned int uiR_data;

    if(*P_SPI_RX_STATUS & C_SPI_RX_FLAG)          // 是否有接收中断
    {
        uiR_data = *P_SPI_RX_DATA;                // 接收数据
    }
    *P_SPI_RX_STATUS |= C_SPI_RX_FLAG;            // 清除接收中断
}
```

4.9.8 注意事项

使用 SPCE3200 的 SPI 接口时需要注意：SPCE3200 作为主机/从机时 SPI_CSN 和 SPI_CLK 的输入/输出方向不同，作为主机时这两个管脚都为输出，作为从机时这两个管脚都为输入。

4.10 I2C

4.10.1 概述

SPCE3200 有一个标准的硬件 I2C（Inter-Integrated Circuit，另外，为了描述方便，本书中所有的 I^2C 都用 I2C 表示）接口，可以方便的与带有 I2C 总线的芯片通讯，可调整总线的通讯时钟频率。

4.10.2 特性

I2C 的特性如下：

- 可作为标准的 I2C 接口；
- 可配置为主机；
- 可编程时钟频率来控制通讯速率；
- 主机、从机之间可实现双向数据传输；
- 握手机制使得主从机之间的通讯更加灵活；
- 支持单字节（8bit）、半字（16bit）、多字节（8bit×n）的数据传输；
- 传输时序中数据地址的传输，使得 SPCE3200 和外部串行存储器之间的读写更加便利。

4.10.3 引脚描述

SPCE3200 有两个 I2C 引脚，如表 4.99 所示：

表 4.99 I2C 引脚列表

引脚名称	引脚号	引脚属性	引脚功能
I2C_CLK	35	O	I2C 时钟引脚

I2C_DATA	36	I/O	I2C 数据引脚

使用 SPCE3200 作为 I2C 总线的主机时，在 SPCE3200 的 I2C_CLK 和 I2C_DATA 接入总线时需要各自接一个 1~10KΩ 的上拉电阻。

结构框图

SPCE3200 的 I2C 接口结构框图如图 4.45：主要包括时钟发生器、控制器和移位寄存器三大部分。

时钟发生器即分频计数器，I2C 的系统时钟（27MHz）经 4 分频，通过分频计数器后提供 I2C 的数据传输时钟频率。

控制器包括开启/传输模式等控制模块、数据传输位数仲裁器和中断控制器，开启/传输模式等控制模块（图中的 I2C 控制器）负责开始或者停止数据传输，选择传输模式；数据传输位数仲裁器控制进入/移出移位寄存器的数据位数；中断控制器负责使能 I2C 中断，并在主机向总线发送起始信号和地址时发出中断请求，发出后中断标志位置 1。

移位寄存器主要负责按照时序控制逻辑单元的控制逻辑把地址、数据及应答信号按位移出到接口发送或者给接收数据单元保存接收数据。

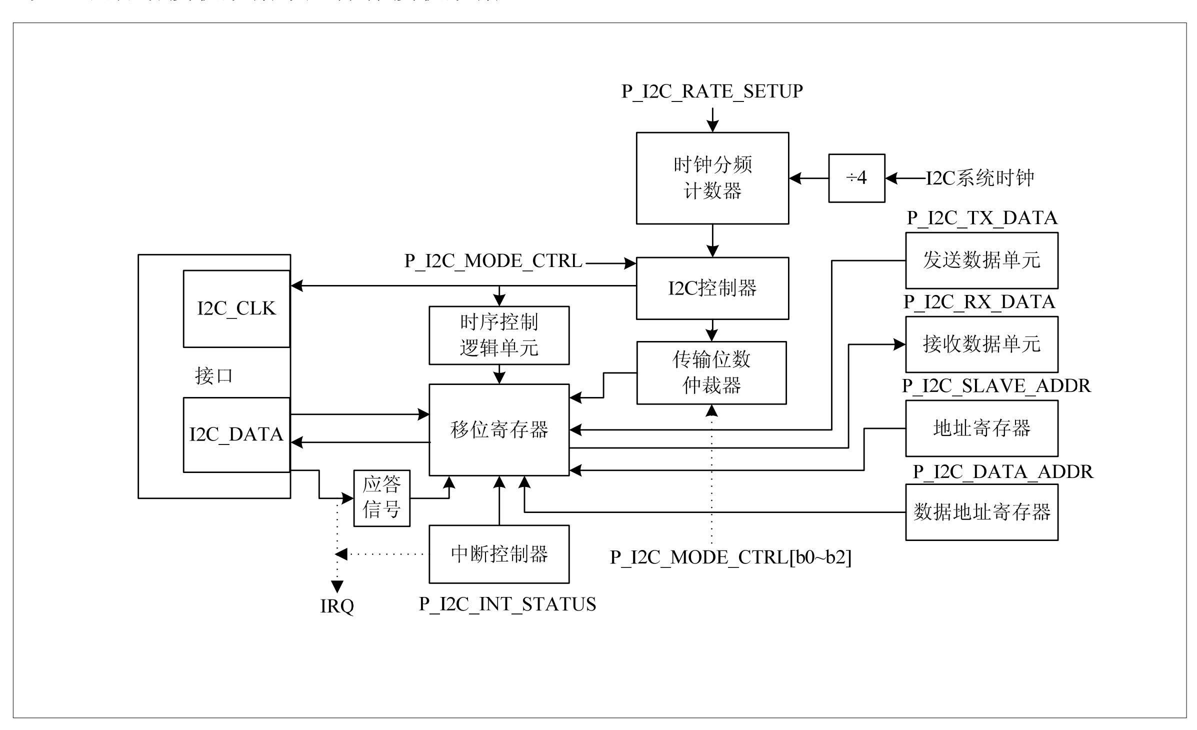


图 4.45 I2C 结构框图

4.10.4 I2C 描述

1、总线配置

I2C 是一个半双工的串行通讯总线，是两线制（数据线和时钟线）总线；一个 I2C 总线上可以连接多个主机和多个从机，由握手信号决定主从机之间的通讯；主机产生所有串行时钟脉冲及起始和结束条件。

SPCE3200 只可以作为 I2C 总线的主机。对于主机来说，按照数据传输的方向可以分为以下两



类总线配置模式：

- 主发送器模式：即主机向从机发送数据。
- 主接收器模式：即从机向主机发送数据。

图 4.46 为 I2C 总线的配置电路。其中 Rp 为上拉电阻，阻值在 1~10KΩ 之间即可。

图 4.47 为 SPCE3200 作为 I2C 总线主机的典型应用电路。

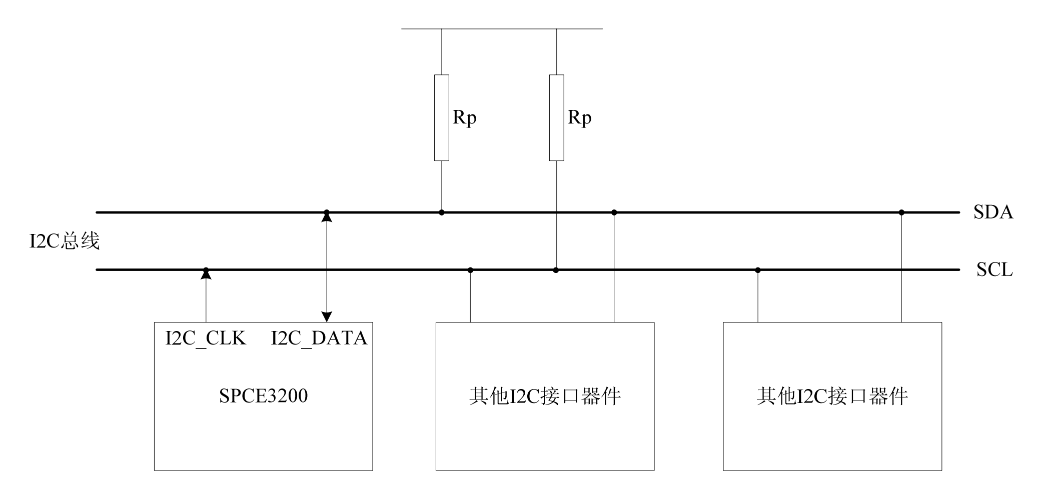


图 4.46 I2C 总线配置

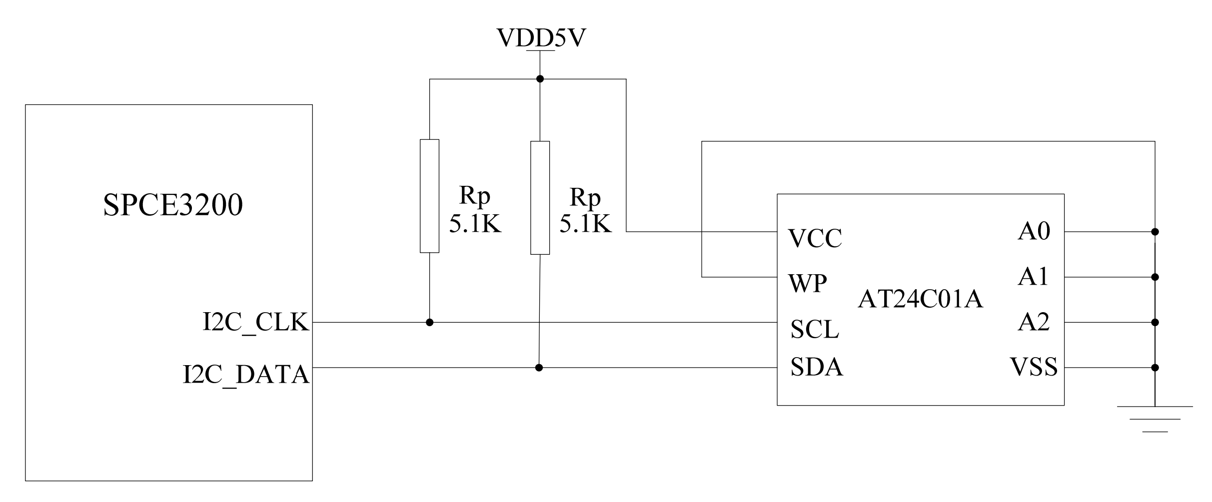


图 4.47 SPCE3200 作为 I2C 总线主机典型电路

2、I2C 数据传输

SPCE3200 的 I2C 数据传输包括主发送和主接收：

（1）主发送

SPCE3200 的主发送是指 SPCE3200 作为主机向 I2C 总线上的从机发送数据，此时 SPCE3200 称作主发送器，因此主发送也被称作主发送器模式。由于 SPCE3200 的 I2C 传输支持单字节、半字和多字节传输，所以主发送器模式又可分别：单字节发送、半字发送和多字节发送模式。

A、单字节主发送模式

单字节主发送模式的数据格式如图 4.48，起始（S）和结束（P）条件均由主机 SPCE3200 产生，分别用于指示串行数据传输开始和结束。在此模式下数据传输模式为发送，所以寄存器

P_I2C_MODE_CTRL（详见第 4.10.5 节）的 bit6 应该被设置为 0，表示执行发送操作。完成发送后，I2C 中断标志置 1。

注意图中从机 ID 地址（8 位）为 7 位从机 ID 地址+1 位 0。

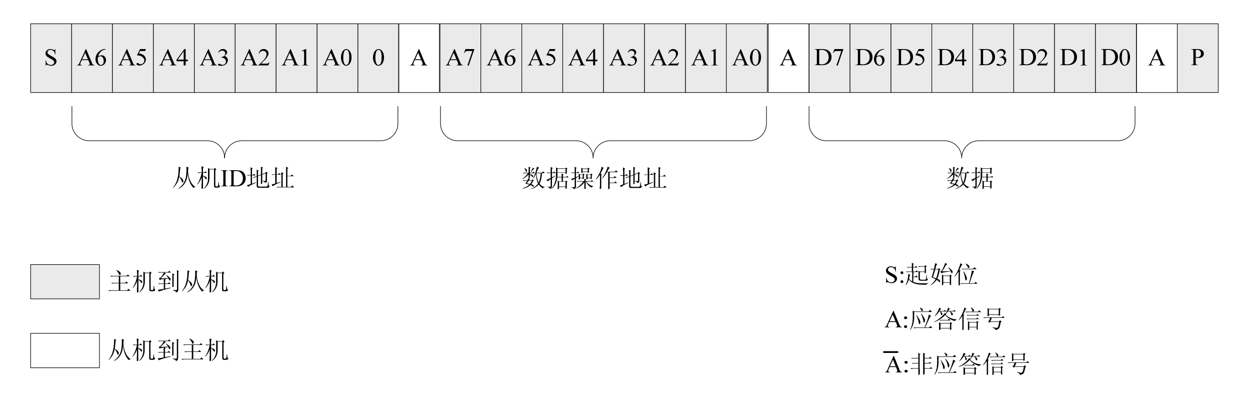


图 4.48 单字节主发送模式的数据格式

单字节主发送模式的时序图如图 4.49：在 SPCE3200 向 I2C 总线上某一从机发送数据时，SPCE3200（主机）需先向 I2C 总线发送一个起始位（起始条件），接着发送对方（从机）的 ID 地址和对从机操作的地址（数据操作地址，如果向一个 I2C 接口存储器 0x00 地址写一个数据，该地址即为 0x00）；如果从机收到这些数据且确认准备好通讯，向主机发送一个应答信号；主机收到应答信号后引发中断开始向从机发送一个字节数据；从机收到一字节数据后向主机发送应答信号；I2C 中断标志置 1；主机发送停止位，一个字节的数据发送完成。

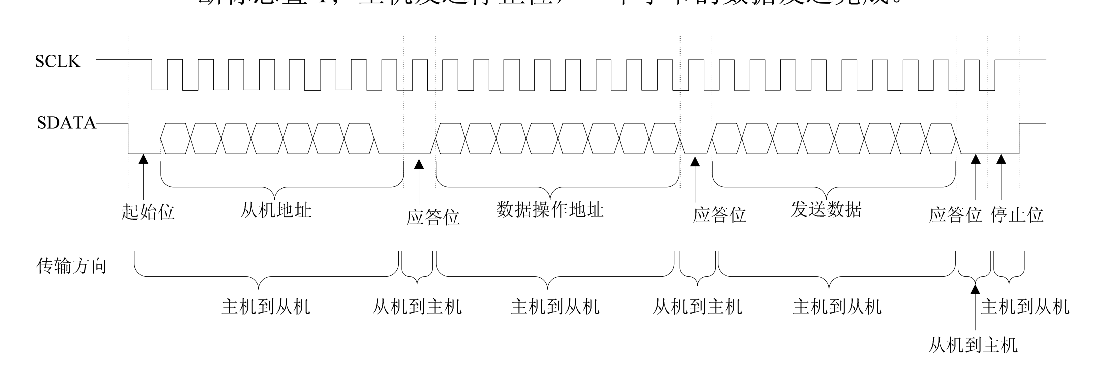


图 4.49 单字节主发送模式的时序图

B、半字主发送模式

半字主发送模式的数据格式如图 4.50，和单字节发送模式类似，所不同的是半字主发送模式的数据位为 16 位（图中 D15~D0）。时序图如图 4.51。

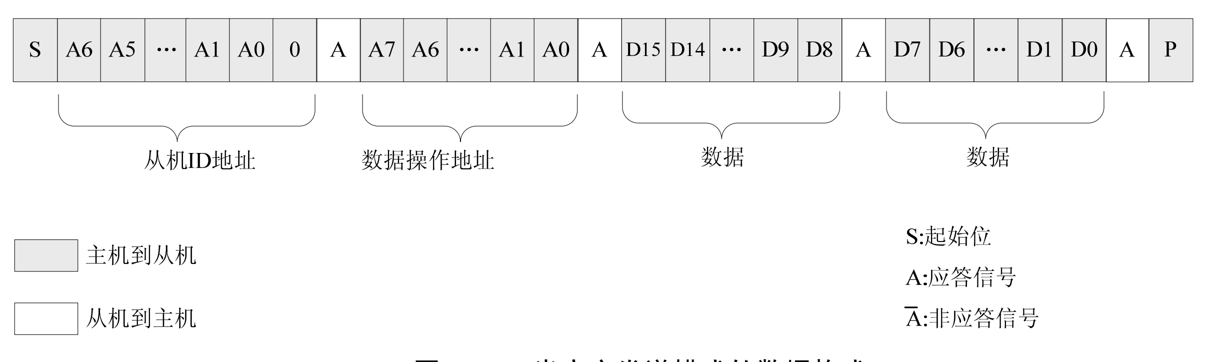


图 4.50 半字主发送模式的数据格式

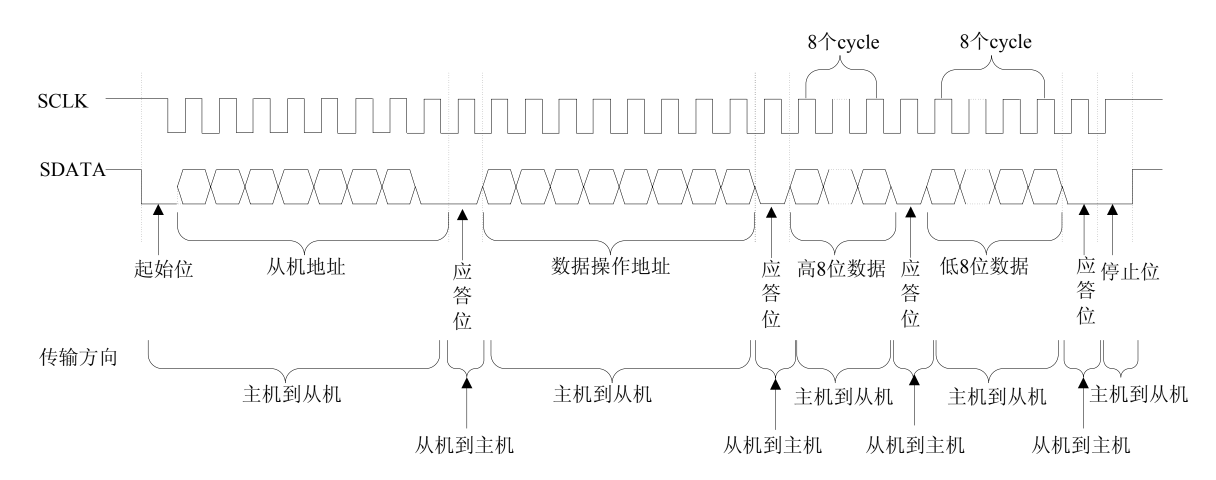


图 4.51　半字主发送模式的时序图

C、多字节发送模式

多字节（8×N）发送模式的数据格式如图 4.52，多字节发送和单字节或者半字发送的过程不同，单字节、半字是在完成所有数据（所有一次传输中包含的 ID 地址、数据操作地址、数据和应答）的传输后触发中断，而多字节传输时每传输完 8 位数据就会触发一次中断，例如，发送完从机 ID 地址，触发一次中断，发送完数据操作地址，又会触发一次中断，依次类推。所以，多字节传输模式的最小传输单位是 8 位，也称为一个 Phase。

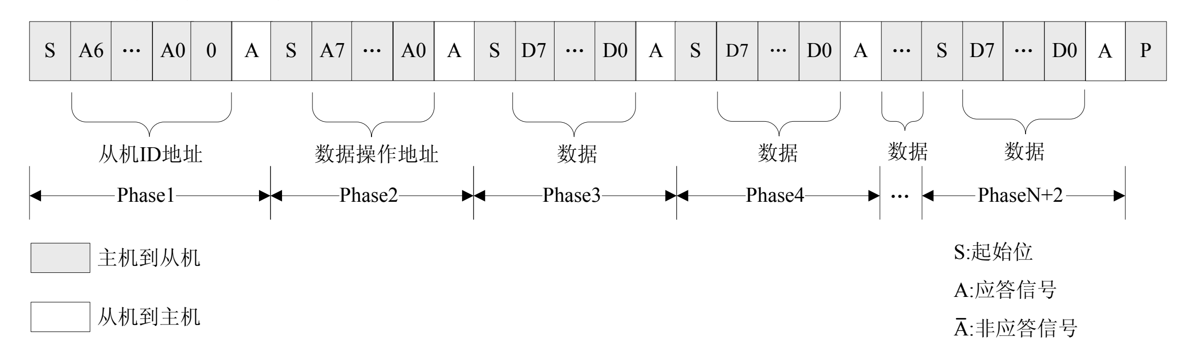


图 4.52　多字节主发送模式的数据格式

一个 Phase 的数据发送时序图如图 4.53。图 4.52 中每一个 Phase 都可以看作一次独立的数据发送，N 个字节的数据的发送可以看作发送 1 个 Phase 的从机地址、发送 1 个 Phase 的数据操作地址和 N 个 Phase 的数据的过程。

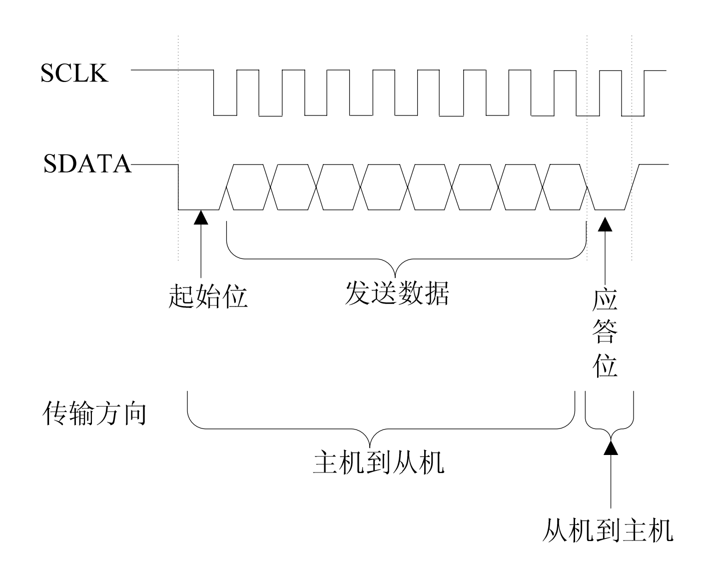


图 4.53　一个 Phase 的数据发送时序图

（2）主接收

SPCE3200 的主接收是指 SPCE3200 作为主机接收来自 I2C 总线上从机的数据，此时 SPCE3200 称作主接收器，因此主接收也被称作主接收器模式。同样由于 SPCE3200 的 I2C 传输支持单字节、半字和多字节传输，所以主接收器模式又可分别：单字节接收、半字接收和多字节接收模式。

A、单字节主接收模式

单字节主接收模式的数据格式如图 4.54，起始（S）和结束（P）条件均由主机 SPCE3200 产生，分别用于指示串行数据传输开始和结束。在此模式下数据传输模式为接收，所以寄存器 P_I2C_MODE_CTRL（详见第 4.10.5 节）的 bit6 应该被设置为 1，表示执行接收操作。

注意图中出现了两个从机 ID 地址（8 位），第一个从机 ID 地址为 7 位从机 ID 地址+1 位“0”；第二个从机 ID 地址为 7 位从机 ID 地址+1 位“1”。两个从机 ID 地址的功能不同：第一个从机 ID 地址为一个握手信号，说明和地址 A7~A0 的从机进行通讯；第二个从机 ID 地址之后带一位“1”，表示要进行的操作为读操作。

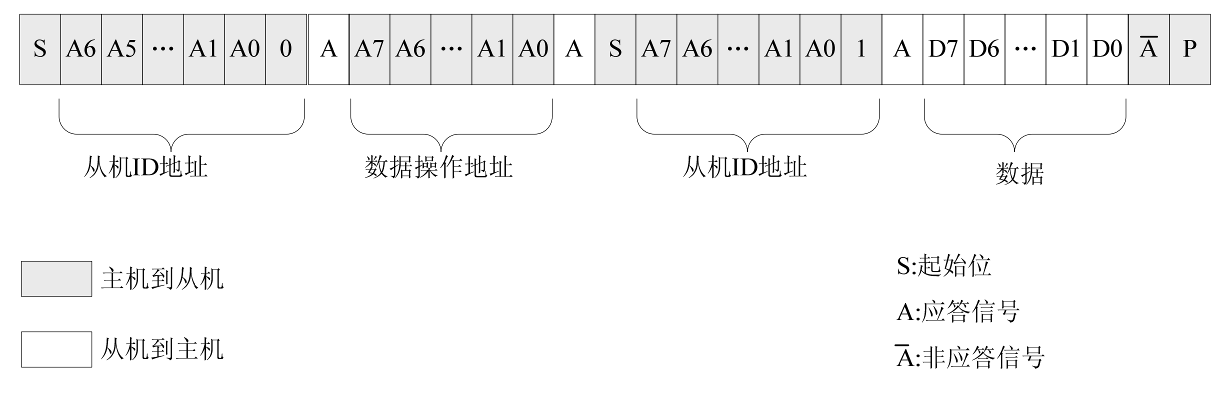


图 4.54　单字节主接收模式的数据格式

单字节主接收模式的时序图如图 4.55：SPCE3200（主机）需先向 I2C 总线发送一个起始位（起始条件），接着发送对方（从机）的 ID 地址和对从机操作的地址（数据操作地址，如果从一个 I2C 接口存储器 0x00 地址读一个数据，该地址即为 0x00），并发送读写条件（低电平表示写，高电平表示读）；如果从机收到这些数据且确认准备好状态，向主机发送一个应答信号；主机收到应答信号后发送从机地址，注意从机地址的最低位为 1；从机接收到该信息后，给主机发送应答信号，并发送

一字节的数据，发送完成主机接收到数据后，主机返回一非应答信号，表示一字节数据接收完成；I2C 中断标志置 1；主机发送停止位，一个字节的数据接收结束。

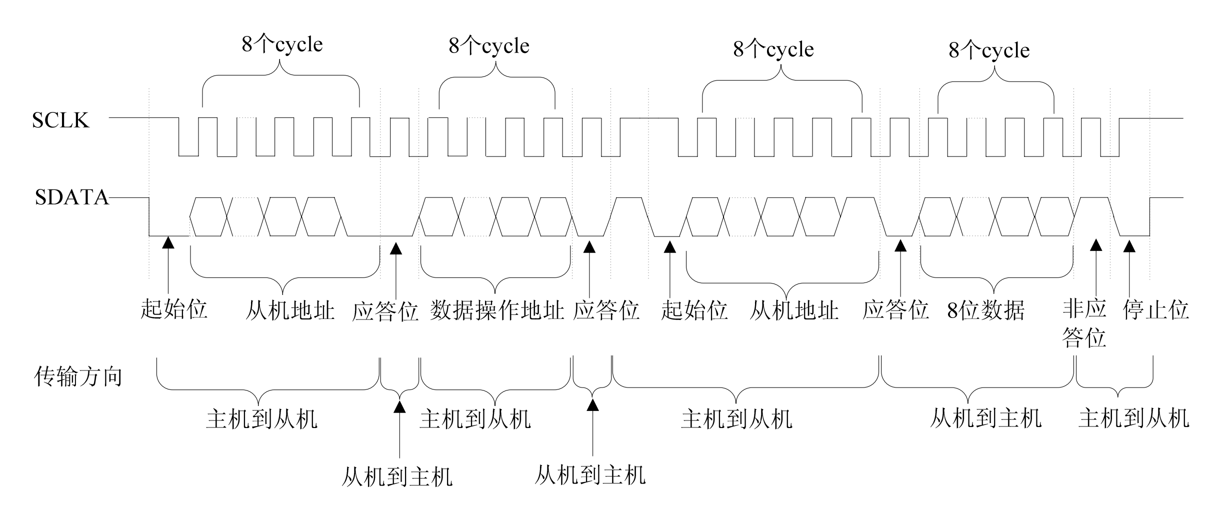


图 4.55　单字节主接收模式的时序图

（2）半字主接收模式

半字主接收模式的数据格式如图 4.56，和单字节接收模式类似，所不同的是半字主接收模式的数据位为 16 位（图中 D15~D0）。时序图如图 4.57。

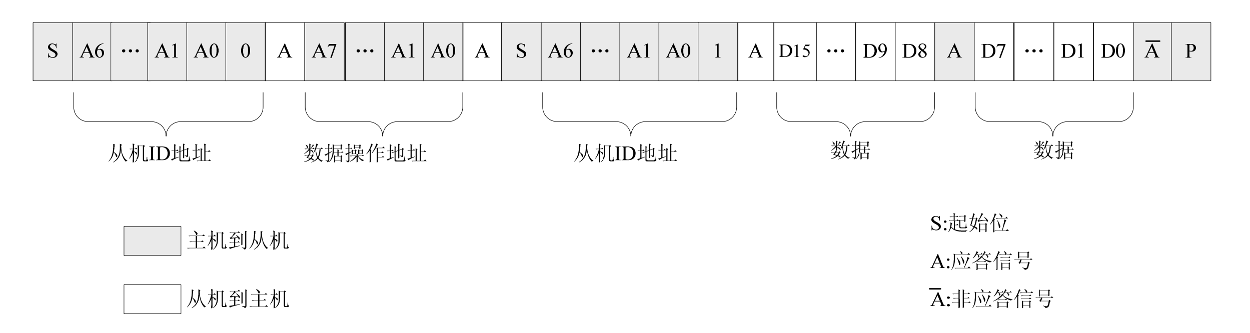


图 4.56　半字主接收模式的数据格式

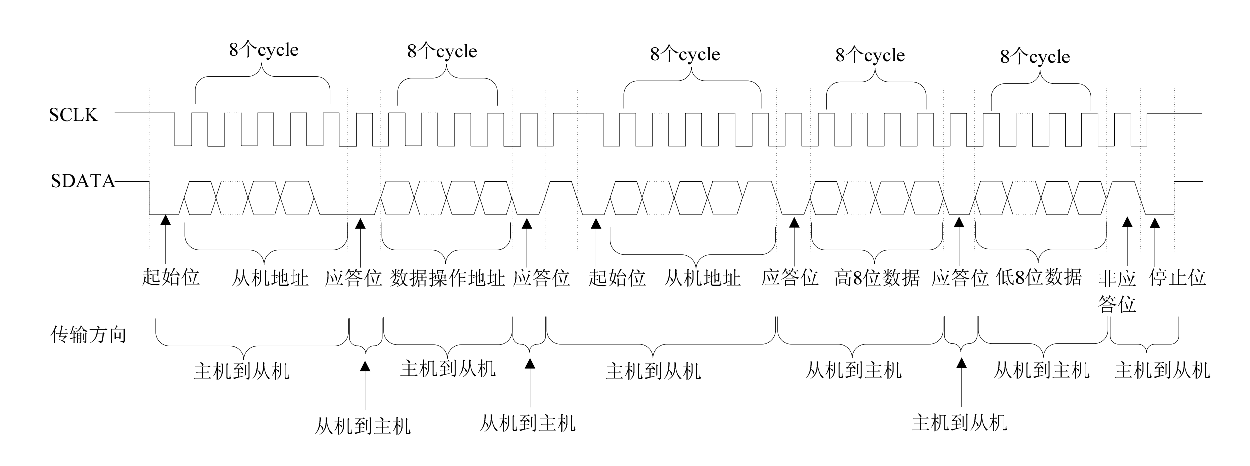


图 4.57　半字主接收模式的时序图

（3）多字节主接收模式

多字节（8×N）主接收模式的数据格式如图 4.58，多字节接收和单字节或者半字接收的过程不同，后两者是在完成所有数据（所有一次传输中包含的 ID 地址、数据操作地址、数据和应答）的传输后触发中断，而多字节传输时每传输完 8 位数据就会触发一次中断，例如，发送完从机 ID 地址，触发一次中断，发送完数据操作地址，又会触发一次中断，依次类推。所以，多字节传输模式的最小传输单位是 8 位，同样称为一个 Phase。

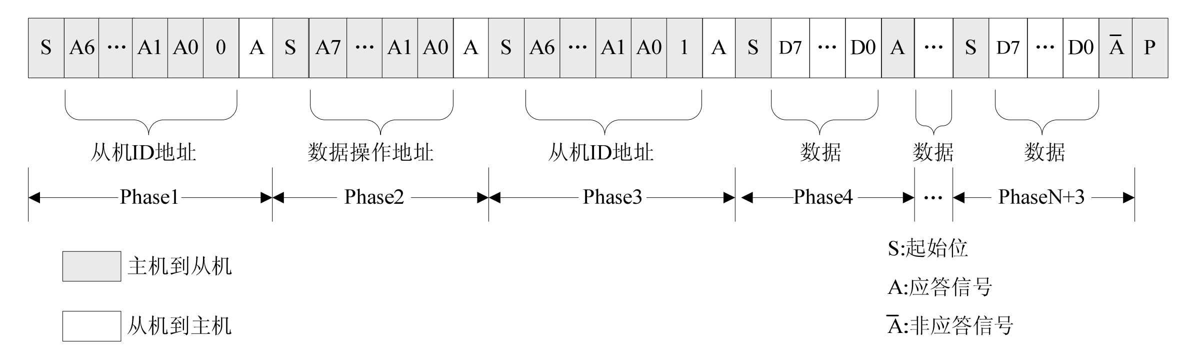


图 4.58　多字节主接收模式的数据格式

一个 Phase 的数据接收时序图如图 4.59，一个 Phase 的数据发送时序图如图 4.53。图 4.58 中每一个 Phase 都可以看作一次独立的数据传输，N 个字节的数据的接收可以看作发送 1 个 Phase 的从机地址、发送 1 个 Phase 的数据操作地址、再发送 1 个 Phase 的从机地址（最低位为 1）和接收 N 个 Phase 的数据的过程。

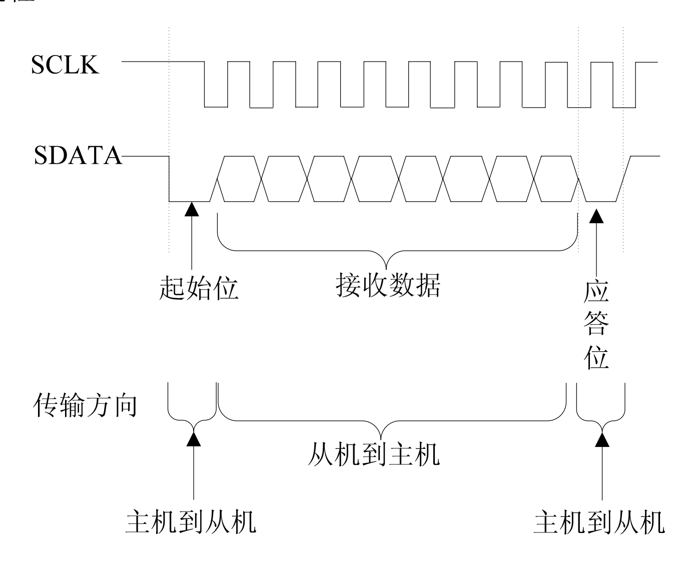


图 4.59　一个 Phase 的数据接收时序图

3、I2C 的应用

I2C 总线具有连接线少（只有两线），连接简单和通讯灵活、方便等特点，主要用于多设备、短距离通讯的场合，例如可应用于：

- 串行存储器，如 E2PROM、RAM 等；
- 串行外设，如 LCD 控制器、时钟芯片和音调发生器等。



4.10.5 寄存器描述

I2C 模块相关的寄存器共有 12 个，如表 4.100。

表 4.100 I2C 相关寄存器列表

寄存器名称	助记符	地址
I2C GPIO 设置寄存器	P_I2C_GPIO_SETUP	0x88200044
I2C GPIO 输入数据寄存器	P_I2C_GPIO_INPUT	0x88200074
I2C GPIO 外部中断寄存器	P_I2C_GPIO_INT	0x88200098
I2C 接口选择寄存器	P_I2C_INTERFACE_SEL	0x88200004
I2C 时钟配置寄存器	P_I2C_CLK_CONF	0x88210094
I2C 控制寄存器	P_I2C_MODE_CTRL	0x88130020
I2C 中断状态寄存器	P_I2C_INT_STATUS	0x88130024
I2C 传输速率设置寄存器	P_I2C_RATE_SETUP	0x88130028
I2C 从机地址寄存器	P_I2C_SLAVE_ADDR	0x8813002C
I2C 数据地址寄存器	P_I2C_DATA_ADDR	0x88130030
I2C 发送数据寄存器	P_I2C_TX_DATA	0x88130034
I2C 接收数据寄存器	P_I2C_RX_DATA	0x88130038

如果 I2C 接口复用为 GPIO 功能，可以对表 4.100 的前三个寄存器操作。寄存器的各位对应引脚关系如表 4.142：

表 4.101 UART 接口复用为 GPIO 引脚与寄存器位对应关系

引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
I2C_CLK	35	bit[0]	可以
I2C_DATA	36	bit[1]	可以

I2C GPIO 设置寄存器：P_I2C_GPIO_SETUP(0x88200044)

I2C GPIO 设置寄存器的功能是：I2C 复用 GPIO 使用时，设置输出使能、上/下拉电阻输入和输出数据。

表 4.102 P_I2C_GPIO_SETUP(0x88200044)

位	b31~b26	b25~b24	b23~b18	b17~b16	b15~b10	b9~b8	b7~b2	b1~b0
读/写	-	W	-	W	-	W	-	W
默认值	-	0x3	-	0	-	0	-	0
名称	-	XI2C_PD	-	XI2C_PU	-	I2C_OE	-	I2C_O

XI2C_PD	b25~b24	I2C GPIO 下拉使能位： 0：不使能为下拉口 1：使能为下拉口
XI2C_PU	b17~b16	I2C GPIO 上拉使能位： 0：使能为上拉口 1：不使能为上拉口
I2C_OE	b9~b8	I2C GPIO 输出使能位： 0：不使能位输出口 1：使能为输出口
I2C_O	b1~b0	I2C GPIO 输出数据

注意：b0、b8、b16、b24 控制 I2C_CLK 管脚；b1、b9、b17、b25 控制 I2C_DATA 管脚。

I2C GPIO 输入数据寄存器：P_I2C_GPIO_INPUT(0x88200074)

I2C GPIO 输入数据寄存器各位功能如下：

表 4.103　P_I2C_GPIO_INPUT(0x88200074)

位	b31~b26	b25~b24	b23~b0
读/写	-	R	-
默认值	-	0	-
名称	-	I2C_IN	-

I2C_IN	b25~b24	I2C 端口作为 GPIO 的输入数据

I2C GPIO 外部中断寄存器：P_I2C_GPIO_INT(0x88200098)

I2C GPIO 外部中断寄存器各位功能如下：

表 4.104　P_I2C_GPIO_INPUT(0x88200074)

位	b31~b26	b25~b24	b23~b18	b17~b16	b15~b10	b9~b8	b7~b2	b1~b0
读/写	-	R/W	-	R/W	-	W	-	W
默认值	-	0	-	0	-	0	-	0
名称	-	I2C_FI	-	I2C_RI	-	I2C_FIEN	-	I2C_RIEN

I2C_FI	b25~b24	I2C GPIO 下降沿中断标志位： 读 0：没有发生下降沿中断

读 1：发生下降沿中断
写 0：无意义
写 1：清除中断标志

I2C_RI　b17~b16　I2C GPIO 上升沿中断标志位：
读 0：没有发生上升沿中断
读 1：发生上升沿中断
写 0：无意义
写 1：清除中断标志

I2C_FIEN　b9~b8　I2C GPIO 下降沿中断使能位：
0：不使能下降沿中断
1：使能下降沿中断

I2C_RIEN　b1~b0　I2C GPIO 上升沿中断使能位：
0：不使能上升沿中断
1：使能上升沿中断

I2C 接口选择寄存器：P_I2C_INTERFACE_SEL(0x88200004)

通过 I2C 接口选择寄存器选择 I2C 接口的复用功能。

表 4.105　P_I2C_INTERFACE_SEL(0x88200004)

位	b31~b1	b0
读/写	-	R/W
默认值	-	0
名称	-	SW_I2C

SW_I2C　b0　I2C 复用端口功能选择位：
0：作为 GPIO
1：作为 I2C 接口

I2C 时钟配置寄存器：P_I2C_CLK_CONF(0x88210094)

通过 I2C 时钟配置寄存器可以停止/打开 I2C 时钟或者复位/不复位 I2C 模块，在使能 I2C 前需先打开 I2C 时钟。

表 4.106　P_I2C_CLK_CONF(0x88210094)

位	b31~b2	b1	b0
读/写	-	W	W
默认值	-	1	0

名称	-	I2C_RST	I2C _STOP

I2C _RST　　b1　　I2C 模块时钟复位位：

0：I2C 模块时钟复位

1：I2C 模块时钟不复位

I2C _STOP　　b0　　I2C 模块时钟使能位：

0：I2C 模块时钟停止

1：I2C 模块时钟使能

I2C 传输速率设置寄存器：P_I2C_RATE_SETUP(0x88130028)

通过 I2C 传输速率设置寄存器可以设置其传输速率。

表 4.107　P_I2C_RATE_SETUP(0x88130028)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	CLK_SET

CLK_SET　b9~b0　　I2C 时钟频率设置位。I2C 时钟频率=27M/4/CLK_SET。

I2C 控制寄存器：P_I2C_MODE_CTRL(0x88130020)

I2C 控制寄存器用来控制开始/停止数据传输，选择读（接收）/写（发送）模式，选择传输的数据位数等。

表 4.108　P_I2C_MODE_CTRL(0x88130020)

位	b8	b7	b6	b5	b4	b3	b2	b1	b0
读/写	W	W	W	R	R	R	W	W	W
默认值	0	0	0	0	0	0	0	0	0
名称	STOPN	GAK	MODE	ACKN	ACK16	ACK8	STARTN	START16	START8

STOPN　　b8　　停止 8×n 位数据传输使能位：

0：不停止 8×n 位数据传输

1：停止 8×n 位数据传输

GAK　　b7　　传输结束时 ACK/NAK 应答信号位：

0：以 ACK 结束

1：以 NAK（非 ACK）结束

MODE　　b6　　发送器/接收器模式选择：

0：选择为发送器模式

1：选择为接收器模式

ACKN　　b5　　普通数据传输应答位：

0：无应答

1： 8×n 位数据传输应答

ACK16　　b4　　16 位（半字）数据传输应答位：

0：无应答

1：16 位（半字）数据传输应答

ACK8　　b3　　8 位（单字节）数据传输应答位：

0：无应答

1：8 位（单字节）数据传输应答

STARTN　　b2　　普通数据传输起始位：

0：无操作

1：开始 8×n 位数据传输

START16　　b1　　16 位（半字）数据传输起始位：

0：无操作

1：开始 16 位（半字）的数据传输

START8　　b0　　8 位（单字节）数据传输起始位：

0：无操作

1：开始 8 位（单字节）的数据传输

I2C 中断状态寄存器：P_I2C_INT_STATUS(0x88130024)

I2C 中断状态寄存器各位功能如下：

表 4.109　P_I2C_INT_STATUS(0x88130024)

位	b1	b0
读/写	W	R/W
默认值	0	0
名称	INT_EN	INT_FLAG

INT_EN　　b1　　I2C 发送/接收中断使能位：

0：屏蔽发送/接收中断

1：使能发送/接收中断

INT_FLAG　b0　　I2C 发送/接收中断标志位：
读 0：没有发生 I2C 发送/接收中断
读 1：发生 I2C 发送/接收中断
写 0：无意义
写 1：清中断标志

I2C 从机地址寄存器 P_I2C_SLAVE_ADDR(0x8813002C)

I2C 从机地址寄存器各位功能如下：

表 4.110　P_I2C_SLAVE_ADDR(0x8813002C)

位	b31~b8	b7~b1	b0
读/写	-	W	-
默认值	-	0	-
名称	-	ID_REG	-

ID_REG　b7~b1　　I2C 从机地址，共 7 位

I2C 数据地址寄存器：P_I2C_DATA_ADDR(0x88130030)

I2C 数据地址寄存器用来指定操作数据的源地址，即源数据在从机设备中的地址，例如外部 RAM 中的存储数据地址等。

表 4.111　P_I2C_DATA_ADDR(0x88130030)

位	b31~b8	b7~b0
读/写	-	W
默认值	-	0
名称	-	ADDR

ID_REG　b7~b0　　I2C 目标地址

I2C 发送数据寄存器：P_I2C_TX_DATA(0x88130034)

I2C 发送数据寄存器各位功能如下：

表 4.112　P_I2C_TX_DATA(0x88130034)

位	b31~b16	b15~b0
读/写	-	W
默认值	-	0

名称	-	I2C_WDATA

I2C_WDATA　b15~b0　I2C 准备发送的数据。

I2C 接收数据寄存器：P_I2C_RX_DATA(0x88130038)

I2C 接收数据寄存器各位功能如下：

表 4.113　P_I2C_RX_DATA(0x88130038)

位	b15~b0
读/写	R
默认值	0
名称	I2C_RDATA

I2C_RDATA　b15~b0　I2C 接收到的数据。

4.10.6 基本操作

SPCE3200 可以作为 I2C 总线的主机，可以和 I2C 总线上的从机进行数据传输，包括数据发送和接收，因此 SPCE3200 的 I2C 通讯包括主发送和主接收。

1、 8bit 模式发送与接收

参考程序段如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
// 8bit  发送与接收
int main(void)
{
        unsigned int i = 0;                                    // 软件延时使用
        unsigned int j = 0;                                    // 发送的数据与地址
        unsigned int uiData = 0;                               // 接收数据

        *P_I2C_CLK_CONF = C_I2C_CLK_EN
                          | C_I2C_RST_DIS;                     // 使能 I2C 模块时钟
        *P_I2C_INTERFACE_SEL = C_I2C_PORT_SEL;                 // 端口选择 I2C
        *P_I2C_RATE_SETUP = 67;                // 传输速率在 100K，27M/4/67 约为 100K
        *P_I2C_INT_STATUS = C_I2C_INT_EN;                      // I2C 中断使能
        while(1)
```

```
        {
            j++;
            *P_I2C_SLAVE_ADDR = 0xa0;                          // 器件 ID
            *P_I2C_DATA_ADDR = j;                              // 存储地址
            *P_I2C_TX_DATA = j;                                // 写入数据
            *P_I2C_MODE_CTRL = C_I2C_TX_MODE                   // I2C 选择发送模式
                        | C_I2C_8BIT_START;                    // 8bit 模式
            while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));      // 发送
            *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;               // 清除中断
            while(!(*P_I2C_MODE_CTRL & C_I2C_8BIT_ACK));       // 等待应答信号
            *P_I2C_MODE_CTRL = C_I2C_END_ACK;                  // 发送停止位 P，发送结束
            for(i=0;i<10000;i++);                              // 软件延时

            *P_I2C_MODE_CTRL = C_I2C_RX_MODE                   // I2C 选择接收模式
                        | C_I2C_8BIT_START;                    // 8bit 模式
            while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));      // 接收
            *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;               // 清除中断
            uiData = *P_I2C_RX_DATA;                           // 接收数据
            *P_I2C_MODE_CTRL = C_I2C_8BIT_ACK                  // 发送应答信号
                        | C_I2C_END_ACK;                       // 发送停止位 P，接收结束
        }
        return 0;
    }
```

2、 16bit 模式发送与接收

参考程序如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
// 16bit  发送与接收
int main(void)
{
    unsigned int i = 0;                                        // 软件延时使用
    unsigned int j = 0;                                        // 发送的数据与地址
    unsigned int uiData = 0;                                   // 接收数据
    unsigned int uiTdata = 0x1111;                             // 发送的数据
```

```
        *P_I2C_CLK_CONF = C_I2C_CLK_EN
                        | C_I2C_RST_DIS;                    // 使能 I2C 模块时钟
        *P_I2C_INTERFACE_SEL = C_I2C_PORT_SEL;              // 端口选择 I2C
        *P_I2C_RATE_SETUP = 67;                             // 传输速率在 100K，27M/4/67
约为 100K
        *P_I2C_INT_STATUS = C_I2C_INT_EN;                   // I2C 中断使能
        while(1)
        {
            j += 2;
            uiTdata += 1;
            *P_I2C_SLAVE_ADDR = 0xa0;                       // 器件 ID
            *P_I2C_DATA_ADDR = j;                           // 存储地址
            *P_I2C_TX_DATA = uiTdata;                       // 写入数据
            *P_I2C_MODE_CTRL = C_I2C_TX_MODE                // I2C 选择发送模式
                        | C_I2C_16BIT_START;                // 16bit 模式
            while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));   // 发送
            *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;            // 清除中断
            while(!(*P_I2C_MODE_CTRL & C_I2C_16BIT_ACK));   // 等待应答信号
            *P_I2C_MODE_CTRL = C_I2C_END_ACK;               // 发送停止位 P，发送结束
            for(i=0;i<10000;i++);                           // 软件延时

            *P_I2C_MODE_CTRL = C_I2C_RX_MODE                // I2C 选择接收模式
                        | C_I2C_16BIT_START;                // 16bit 模式
            while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));   // 接收
            *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;            // 清除中断
            uiData = *P_I2C_RX_DATA;                        // 接收数据
            *P_I2C_MODE_CTRL = C_I2C_16BIT_ACK              // 发送应答信号
                        | C_I2C_END_ACK;                    // 发送停止位 P，接收结束
        }
        return 0;
    }
```

（1） 多字节数据发送与接收

多字节数据发送的操作相对比较复杂，中断方式和查询方式比较类似，所不同的是中断方式可以在中断服务程序中判断是否返回应答信号，从而决定是否传输下一个数据。这里只介绍比较常用的查询方式。

在介绍多字节数据发送模式时介绍到，多字节数据发送可以看作多个独立 Phase 的数据发送过

程，按照图 4.53 单个 Phase 数据发送的时序图，1 个 Phase 的数据发送流程如**错误！未找到引用源。**。

例：使用 8N bit 模式发送接收 1 个字节，参考程序如下：

```
#include "SPCE3200_Register.h"
#include "SPCE3200_Constant.h"
// 8N bit 模式发送与接收 1 个字节
int main(void)
{
    unsigned int i = 0;                                   // 软件延时使用
    unsigned int j = 0;                                   // 发送的数据与地址
    unsigned int uiData = 0;                              // 接收数据
    unsigned int uiTdata = 0x11;                          // 发送的数据

    *P_I2C_CLK_CONF = C_I2C_CLK_EN
                    | C_I2C_RST_DIS;                      // 使能 I2C 模块时钟
    *P_I2C_INTERFACE_SEL = C_I2C_PORT_SEL;                // 端口选择 I2C
    *P_I2C_RATE_SETUP = 67;                               // 传输速率在 100K，27M/4/27
                                                          // 约为 100K
    *P_I2C_INT_STATUS = C_I2C_INT_EN;                     // I2C 中断使能
    while(1)
    {
        j += 1;                                           // 选择器件地址
        uiTdata += 1;                                     // 改变写入数据
        *P_I2C_TX_DATA = 0xa0;                            // 器件 ID
        *P_I2C_MODE_CTRL = C_I2C_TX_MODE                  // I2C 选择发送模式
                | C_I2C_8NBIT_START;                      // 8N bit  模式
        while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));     // 等待发送
        *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;              // 清除中断
        while(!(*P_I2C_MODE_CTRL & C_I2C_8NBIT_ACK));     // 等待应答信号

        *P_I2C_TX_DATA = j;                               // 数据地址
        *P_I2C_MODE_CTRL = C_I2C_TX_MODE                  // I2C 选择发送模式
                | C_I2C_8NBIT_START;                      // 8N bit  模式
        while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));     // 等待发送
        *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;              // 清除中断
        while(!(*P_I2C_MODE_CTRL & C_I2C_8NBIT_ACK));     // 等待应答信号
```

```
*P_I2C_TX_DATA = uiTdata;                           //  发送的数据
*P_I2C_MODE_CTRL = C_I2C_TX_MODE                    // I2C 选择发送模式
             | C_I2C_8NBIT_START;                   // 8N bit  模式
while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));       //  等待发送
*P_I2C_INT_STATUS |= C_I2C_INT_FLAG;                //  清除中断
while(!(*P_I2C_MODE_CTRL & C_I2C_8NBIT_ACK));       //  等待应答信号

*P_I2C_MODE_CTRL = C_I2C_8NBIT_STOP;                //  发送停止位 P，发送结束
for(i=0;i<10000;i++);                               //  软件延时



*P_I2C_TX_DATA = 0xa0;                              //  器件 ID
*P_I2C_MODE_CTRL = C_I2C_TX_MODE                    // I2C 选择发送模式
             | C_I2C_8NBIT_START;                   // 8N bit  模式
while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));       //  等待发送
*P_I2C_INT_STATUS |= C_I2C_INT_FLAG;                //  清除中断
while(!(*P_I2C_MODE_CTRL & C_I2C_8NBIT_ACK));       //  等待应答信号

*P_I2C_TX_DATA = j;                                 //  数据地址
*P_I2C_MODE_CTRL = C_I2C_TX_MODE                    // I2C 选择发送模式
             | C_I2C_8NBIT_START;                   // 8N bit  模式
while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));       //  等待发送
*P_I2C_INT_STATUS |= C_I2C_INT_FLAG;                //  清除中断
while(!(*P_I2C_MODE_CTRL & C_I2C_8NBIT_ACK));       //  等待应答信号
*P_I2C_MODE_CTRL = C_I2C_8NBIT_STOP;                //  发送停止位 P，发送结束

*P_I2C_TX_DATA = 0xa0 | 1;                      //  器件 ID，最低位为 1 表示读操作
*P_I2C_MODE_CTRL = C_I2C_TX_MODE                    // I2C 选择发送模式
             | C_I2C_8NBIT_START;                   // 8N bit  模式
while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));       //  等待发送
*P_I2C_INT_STATUS |= C_I2C_INT_FLAG;                //  清除中断
while(!(*P_I2C_MODE_CTRL & C_I2C_8NBIT_ACK));       //  等待应答信号

*P_I2C_MODE_CTRL = C_I2C_RX_MODE                    // I2C 选择接收模式
             | C_I2C_8NBIT_START;                   // 8N bit 模式
while(!(*P_I2C_INT_STATUS & C_I2C_INT_FLAG));       //  接收
```

```
        *P_I2C_INT_STATUS |= C_I2C_INT_FLAG;            // 清除中断
        uiData = *P_I2C_RX_DATA;                         // 接收数据
        *P_I2C_MODE_CTRL = C_I2C_8NBIT_STOP;             // 发送停止位 P，发送结束

    }
}
```

4.10.7 注意事项

使用 SPCE3200 的 I2C 模块时要注意下面几点：

- 设置 I2C 时钟频率要考虑通讯另一方（从机）支持的时钟频率；
- SPCE3200 只能作为 I2C 的主机使用。

4.11 SIO 控制器

4.11.1 概述

串行 I/O 接口（SIO）是凌阳科技公司自主产权所有的一种串行接口，用于与其它设备进行通讯。这种串口通过 2 个管脚（SCK 和 SDA）就可以进行数据的发送与接收操作。SIO 可以方便的与凌阳其他具有 SIO 接口的芯片进行通讯，如 SPR 系列存储芯片、带 SIO 接口的其他凌阳单片机等。

SIO 协议由起始位、读/写控制位、地址字、数据字以及停止位组成。图 4.60、图 4.61 分别为 SIO 协议的写和读方式的时序，图中 SCK 为时钟信号，而 SDA 则为双向传输的串行数据信号。若在 SCK 为高电平期间 SDA 完成了由 1 到 0 的跳变，这就是 SIO 协议规定的传输的“起始位”；相反，在 SCK 为高电平期间 SDA 完成了由 0 到 1 的跳变，即为协议规定的传输的“停止位”。除了 SIO 的“起始位/停止位”以外，SDA 只能在 SCK 为低电平时改变其逻辑电平。另外，地址字和数据字的传输是以最高有效位（MSB）开头的；而且，若数据字为 16 位的，则 SIO 首先发送/接收的是低字节数据，然后才是高字节数据。

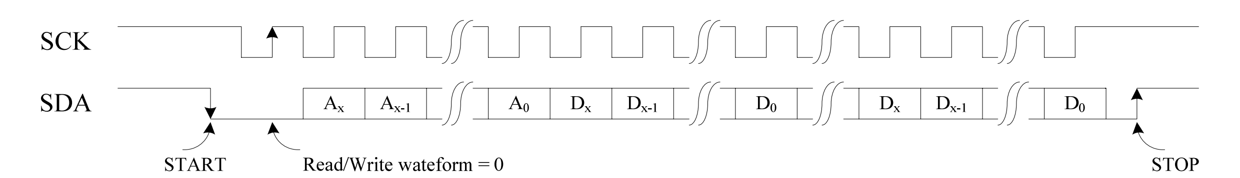


图 4.60 SIO 写操作方式时序

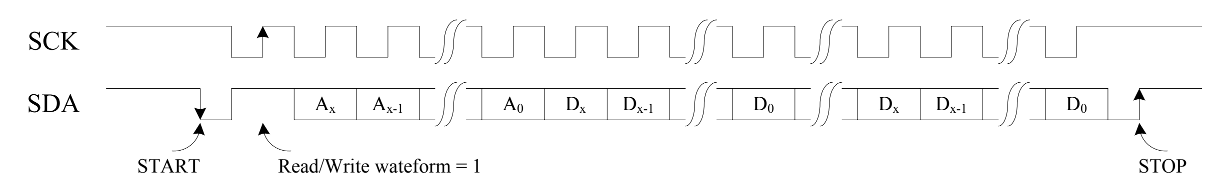


图 4.61 SIO 读操作方式时序

4.11.2 特性

SIO 控制器具有如下一些技术特性：

- 通过 SCK 和 SDA 引脚提供主模读/写功能
- 支持 4 种地址方式：无地址、8 位地址、16 位地址以及 24 位地址
- 支持脉冲读/写信号功能
- 支持 8 位/16 位数据宽度
- 可通过中断/查询两种方式实现数据发送/接收（SIO 控制器使用 IRQ44 中断）
- 支持 4 种 SIO 波特率
- 提供与 SPDS301 芯片通讯的自动位流传输方式

4.11.3 引脚描述

SPCE3200 的 SIO 接口引脚与 SJTAG 和 I2S 接口引脚复用，在使用时需要设置相关寄存器以便使其工作在 SIO 接口方式。SIO 接口的引脚描述如表 4.114 所示。

表 4.114 SIO 接口引脚对应表

引脚名称	引脚号	引脚属性	引脚功能
SIO_RDY	149	I	SIO 接口的状态检测引脚
SIO_SCK	150	O	SIO 接口的时钟输出引脚
SIO_SDA	151	I/O	SIO 接口的数据引脚

4.11.4 结构

SIO 控制器的结构如图 4.62 所示。

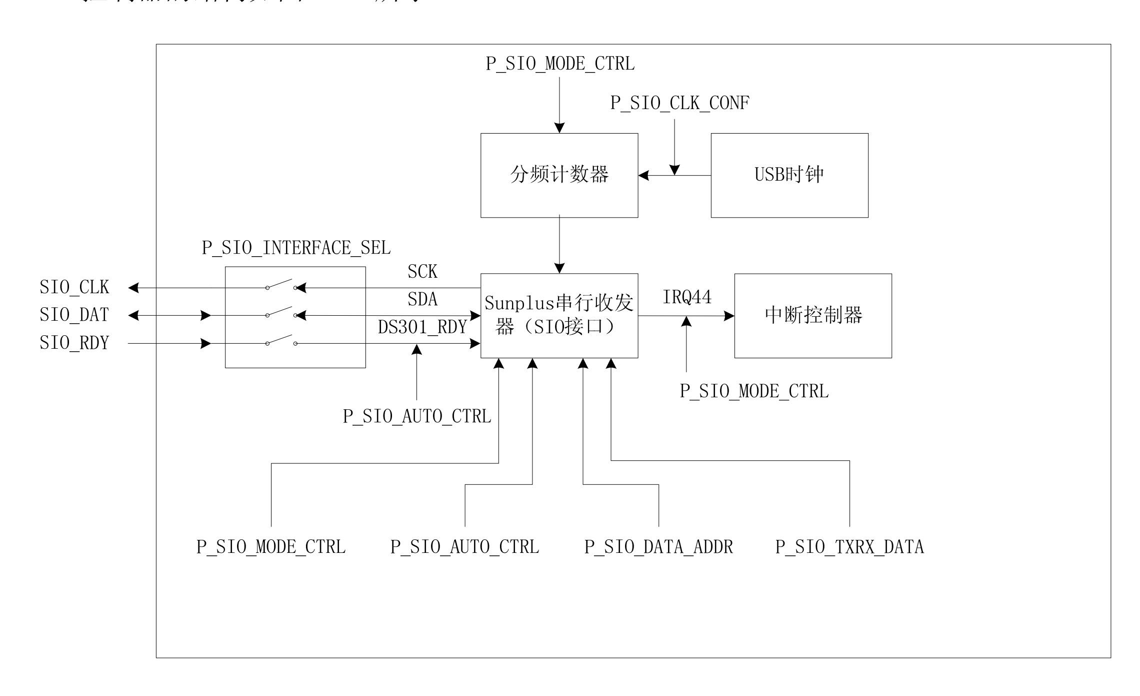


图 4.62 SIO 控制器结构

4.11.5 寄存器描述

SIO 控制器共有 9 个控制寄存器，如表 4.115 所示。通过对这 9 个控制寄存器的操作，即可实

现 SIO 模块的所有功能。

表 4.115 SIO 相关寄存器列表

寄存器名称	助记符	地址
SIO 接口选择寄存器	P_SIO_INTERFACE_SEL	0x88200004
JTAG GPIO 设置寄存器	P_JTAG_GPIO_SETUP	0x88200034
JTAG GPIO 输入数据寄存器	P_JTAG_GPIO_INPUT	0x88200070
JTAG GPIO 外部中断寄存器	P_JTAG_GPIO_INT	0x8820008C
SIO 时钟配置寄存器	P_SIO_CLK_CONF	0x882100A0
SIO 控制寄存器	P_SIO_MODE_CTRL	0X88120000
SIO 自动传输控制寄存器	P_SIO_AUTO_CTRL	0x88120004
SIO 数据地址寄存器	P_SIO_DATA_ADDR	0x88120008
SIO 收发数据寄存器	P_SIO_TXRX_DATA	0x8821000C

如果使用 JTAG 接口（JTAG 模块与 SIO 模块复用引脚）的 GPIO 功能，需要对下面的寄存器操作。寄存器的各位对应引脚关系如：

表 4.116 SIO 接口复用为 GPIO 引脚与寄存器位对应关系

引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
JTAG_TRSTN	153	bit[0]	可以
JTAG_TDI	152	bit[1]	可以
JTAG_TDO	151	bit[2]	可以
JTAG_TCK	150	bit[3]	可以
JTAG_TMS	149	bit[4]	可以

JTAG GPIO 设置寄存器：P_JTAG_GPIO_SETUP(0x88200034)

JTAG GPIO 设置寄存器设置 JTAG 接口复用为 GPIO 时的引脚输出使能、输出数据、上拉电阻输入使能和下拉电阻输入使能。

表 4.117 P_JTAG_GPIO_SETUP(0x88200034)

位	b28~b24	b23~b21	b20~b16	b15~b13	b12~b8	b7~b5	b4~b0
读/写	R/W	-	R/W	-	R/W	-	R/W
默认值	1		1		0	-	0
名称	JTAG_PD		JTAG_PU		JTAG_OE	-	JTAG_O



JTAG_PD　　　b28~b24　　JTAG 接口作为 GPIO 使用时的下拉电阻输入使能位：

0：不使能下拉电阻输入

1：使能下拉电阻输入

JTAG_PU　　　b20~b16　　JTAG 接口作为 GPIO 使用时的上拉电阻输入使能位：

0：使能上拉电阻输入

1：不使能上拉电阻输入

JTAG_OE　　　b12~b8　　JTAG 接口作为 GPIO 使用时的输出使能位：

0：不使能输出

1：使能输出

JTAG_O　　　b4~b0　　JTAG 接口作为 GPIO 使用时的输出数据

JTAG GPIO 输入数据寄存器：P_JTAG_GPIO_INPUT(0x88200070)

表 4.118 P_JTAG_GPIO_INPUT(0x88200070)

位	b31~b5	b4~b0
读/写	-	R
默认值	-	0
名称	-	JTAG_INPUT

JTAG_INPUT　　　b4~b0　　JTAG 接口作为 GPIO 使用时的输入数据

JTAG GPIO 外部中断寄存器：P_JTAG_GPIO_INT(0x8820008C)

JTAG GPIO 外部中断寄存器可进行中断使能、中断触发沿设置、中断标志清除等操作。

表 4.119 P_JTAG_GPIO_INT(0x8820008C)

位	b28~b24	b23~b21	b20~b16	b15~b13	b12~b8	b7~b5	b4~b0
读/写	R/W	-	R/W	-	R/W	-	R/W
默认值	0	-	0	-	0	-	0
名称	JTAG_FI	-	JTAG_RI	-	JTAG_FIEN	-	JTAG_RIEN

JTAG_FI　　　b28~b24　　JTAG GPIO 下降沿中断标志位：

读 0：没有发生下降沿中断

读 1：发生下降沿中断

写 0：无意义

写 1：清除中断标志

JTAG_RI　　　　b20~b16　　JTAG GPIO 上升沿中断标志位：

读 0：没有发生上升沿中断

读 1：发生上升沿中断

写 0：无意义

写 1：清除中断标志

JTAG_FIEN　　　b12~b8　　JTAG GPIO 下降沿中断使能位：

0：不使能下降沿中断

1：使能下降沿中断

JTAG_RIEN　　　b4~b0　　JTAG GPIO 上升沿中断使能位：

0：不使能上升沿中断

1：使能上升沿中断

作为 SIO 控制器需要设置以下寄存器：

SIO 接口选择寄存器：P_SIO_INTERFACE_SEL(0x88200004)

SPCE3200 的 SIO 接口引脚与 SJTAG 和 I2S 接口引脚复用，在使用时需要设置 SIO 接口选择寄存器以便使其工作在 SIO 接口方式。

表 4.120 P_SIO_INTERFACE_SEL(0x88200004)

位	b31~b11	b10~b8	b7~b0
读/写	-	W	-
默认值	-	0	-
名称	-	SW_PERI	-

SW_PERI　　　b10~b8　　SIO、JTAG 和 I2S 复用接口选择位：

000：选择做为 JTAG 端口使用

001：选择做为 SIO 端口使用

010：选择做为 I2S 主模块端口使用

011：选择做为 I2S 从模块端口使用

其他：不作为外设端口使用

SIO 时钟配置寄存器：P_SIO_CLK_CONF(0x882100A0)

表 4.121 P_SIO_CLK_CONF(0x882100A0)

位	b31-b2	b1	b0
读/写	-	R/W	R/W



默认值	-	1	0
名称	-	SIO_RST	SIO_STOP

SIO_RST　　b1　　SIO 模块时钟使能位：
　　0：SIO 模块时钟复位
　　1：SIO 模块时钟使能

SIO_STOP　　b0　　SIO 模块时钟停止位：
　　0：SIO 模块时钟停止
　　1：SIO 模块时钟使能

SIO 控制寄存器：P_SIO_MODE_CTRL(0x88120000)

SIO 控制寄存器用于控制 SIO 收发器的工作模式等。

表 4.122 P_SIO_MODE_CTRL(0x88120000)

位	b31	b30	b15	b14	b13	b12
读/写	R	R	W	W	W	W
默认值	0	0	0	0	0	0
名称	SIO_REQUEST	SIO_IRQ_STS	SIO_DATA_WIDTH	APBDMA_EN	IRQ_EN	PROTOCOL
位	b11~b10	b9~b8	b7	b2	b1	b0
读/写	W	W	W	W	W	W
默认值	0	0	0	0	0	0
名称	BAUDRATE	ADDR_MODE	SIO_IRQ_CLR	SIO_TRANSFER	SIO_RW_CTL	SIO_START

SIO_REQUEST　　b31　　SIO 数据传输请求位（只读），当采用查询方式进行数据传输时，读取该位为 1 时：
在写操作下，表示允许写入数据；
在读操作下，表示接收数据完毕，可以从 P_SIO_TXRX_DATA 寄存器读取接收数据

SIO_IRQ_STS　　b30　　SIO 中断状态位（只读）：
　　读 0：没有发生 SIO 中断
　　读 1：发生 SIO 中断

SIO_DATA_WIDTH　　b15　　SIO 数据位宽控制位：
　　0：8 位数据宽度
　　1：16 位数据宽度

APBDMA_EN	b14	APBDMA 通道使能位： 0：由 CPU 传输数据 1：由 APBDMA 传输数据（详细请参考 APBDMA）
IRQ_EN	b13	SIO 中断使能位： 0：禁止 SIO 中断 1：使能 SIO 中断（SIO 使用 IRQ44 中断向量）
PROTOCOL	b12	SIO 协议读/写位波形控制位： 0：写操作下，读/写位波形为低电平；读操作下，读/写位波形为高电平 1：无论写模式还是读模式，读/写位波形均为低电平
BAUDRATE	b11~b10	SIO 总线波特率选择位： 00：27/16MHz 01：27/4MHz 10：27/8MHz 11：27/32MHz
ADDR_MODE	b9~b8	SIO 地址宽度选择位： 00：16 位地址宽度 01：无地址位 10：8 位地址宽度 11：24 位地址宽度
SIO_IRQ_CLR	b7	SIO 中断标志清除位： 写入 1 即可将 SIO 中断请求标志清除
SIO_TRANSFER	b2	首先传输高/低字节数据选择位： 0：首先传输 SIO 收发数据寄存器的高字节 1：首先传输 SIO 收发数据寄存器的低字节
SIO_RW_CTL	b1	SIO 读/写操作选择位： 0：SIO 读操作 1：SIO 写操作
SIO_START	b0	SIO 启动/停止传输控制位： 0：停止 SIO 数据传输 1：启动 SIO 数据传输

SIO 自动传输控制寄存器：P_SIO_AUTO_CTRL(0x88120004)

SIO 自动传输控制寄存器可以控制 SIO 模块使用中断方式自动传输一定数量的数据而不需要人



工查询干预。

表 4.123 P_SIO_AUTO_CTRL(0x88120004)

位	b7~b4	b3~b2	b1	b0
读/写	W	-	R	W
默认值	15	-	1	0
名称	SIO_WORD_NUM	-	DS301_READY	AUTOMATIC_EN

SIO_WORD_NUM b7~b4 自动传输的字数：

以封包形式传递给 SPDS301 的数据字的字数，其默认值为 15，该值不能被设置成 0，否则将产生不可预知的错误，该参数值仅当 AUTOMATIC_EN 位使能的状态下有效

DS301_READY b1 DS301_Ready 管脚检测使能位：

0：禁止（不使用 DS301_Ready 管脚）

1：使能（使用 DS301_Ready 管脚）

AUTOMATIC_EN b0 自动传输方式（与 SPDS301 通讯）使能控制位：

0：关闭自动传输

1：打开自动传输

打开自动传输方式后，SIO 模块将反复查询反映 SPDS301 状态的寄存器，直至读到该寄存器里“decode work”位和“request ready”位均为 1，此时会产生 SIO 中断，通知用户写一个数据到 P_SIO_TXRX_DATA 寄存器中。写完此数据字后，中断会被自动清除。

在自动传输方式下每发生一次 SIO 中断都要写一个数据到 P_SIO_TXRX_DATA 寄存器中。关于 SPDS301 芯片的详细资料，请参见 SPDS301 编程指南文件。

SIO 数据地址寄存器：P_SIO_DATA_ADDR(0x88120008)

SIO 数据地址寄存器可以设置 SIO 模块对外部设备的寻址地址。

表 4.124 P_SIO_DATA_ADDR(0x88120008)

位	b23~b16	b15~b8	b7~b0
读/写	W	W	W
默认值	0	0	0
名称	SIO_ADDRH	SIO_ADDRM	SIO_ADDRL

SIO_ADDRH b23~b16 SIO 起始地址的高字节部分，仅当 24 位地址模式时使用

SIO_ADDRM	b15~b8	SIO 起始地址的中字节部分，24 位和 16 位地址模式时使用
SIO_ADDRL	b7~b0	SIO 起始地址的低字节部分，24 位、16 位和 8 位地址模式时使用

SIO 收发数据寄存器：P_SIO_TXRX_DATA(0x8812000C)

SIO 收发数据寄存器为 SIO 模块发送/接收数据的缓冲单元。向该单元写入或读出数据，SIO 模块将按照串行方式发送或接收数据。

表 4.125 P_SIO_TXRX_DATA(0x8812000C)

位	**b15~b8**	**b7~b0**
读/写	R/W	R/W
默认值	0	0
名称	SIO_DATAH	SIO_DATAL

SIO_DATAH	b15~b8	SIO 数据高字节，仅当数据位宽设置为 16 位时有效
SIO_DATAL	b7~b0	SIO 数据低字节

4.11.6 基本操作

SIO 模块的操作有三种模式：不使用“自动传输”模式；使用自动传输模式，但不使用 DS301_Ready 引脚；使用自动传输模式，并且使用 DS301_Ready 引脚。下面分别介绍：

不使用“自动传输”模式：

在不使用“自动传输”模式下，用户可以控制 SIO 模块进行数据的串行发送/接收操作。图 4.63 所示的是 SIO 模块运行于“非自动传输”模式下的运行流程。

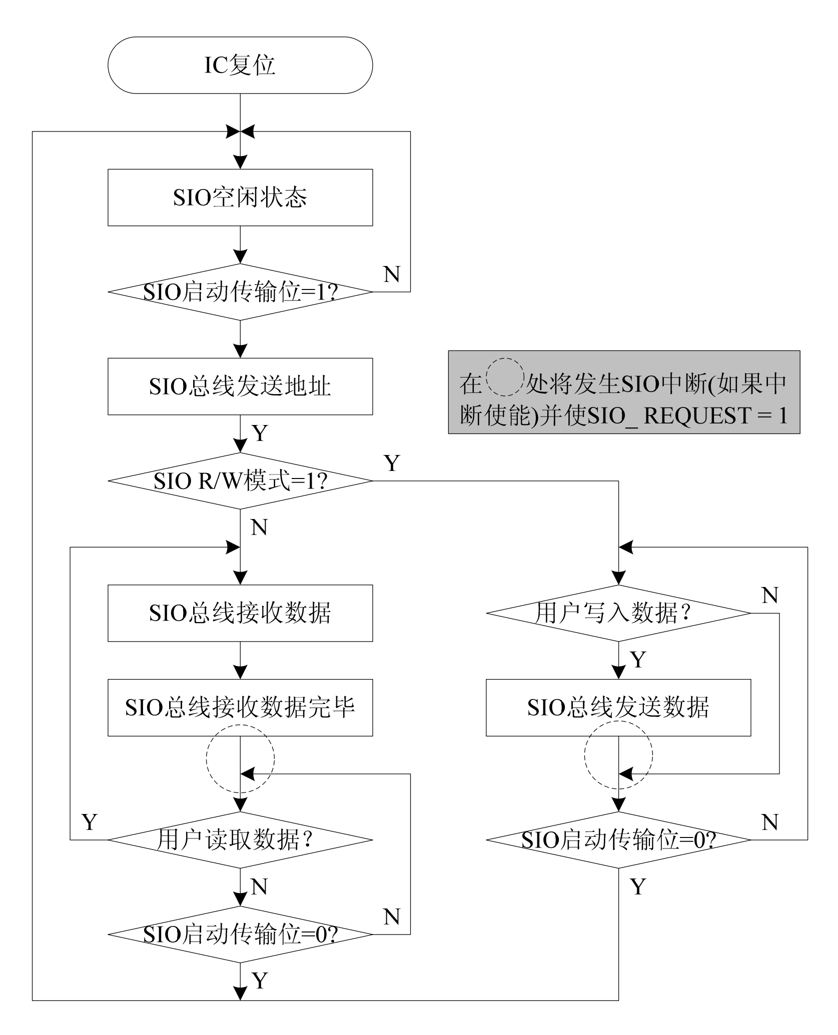


图 4.63 不使用“自动传输”模式传输数据的操作流程

在该模式下使用 SIO 向外设写入数据的操作步骤如图 4.64 所示。

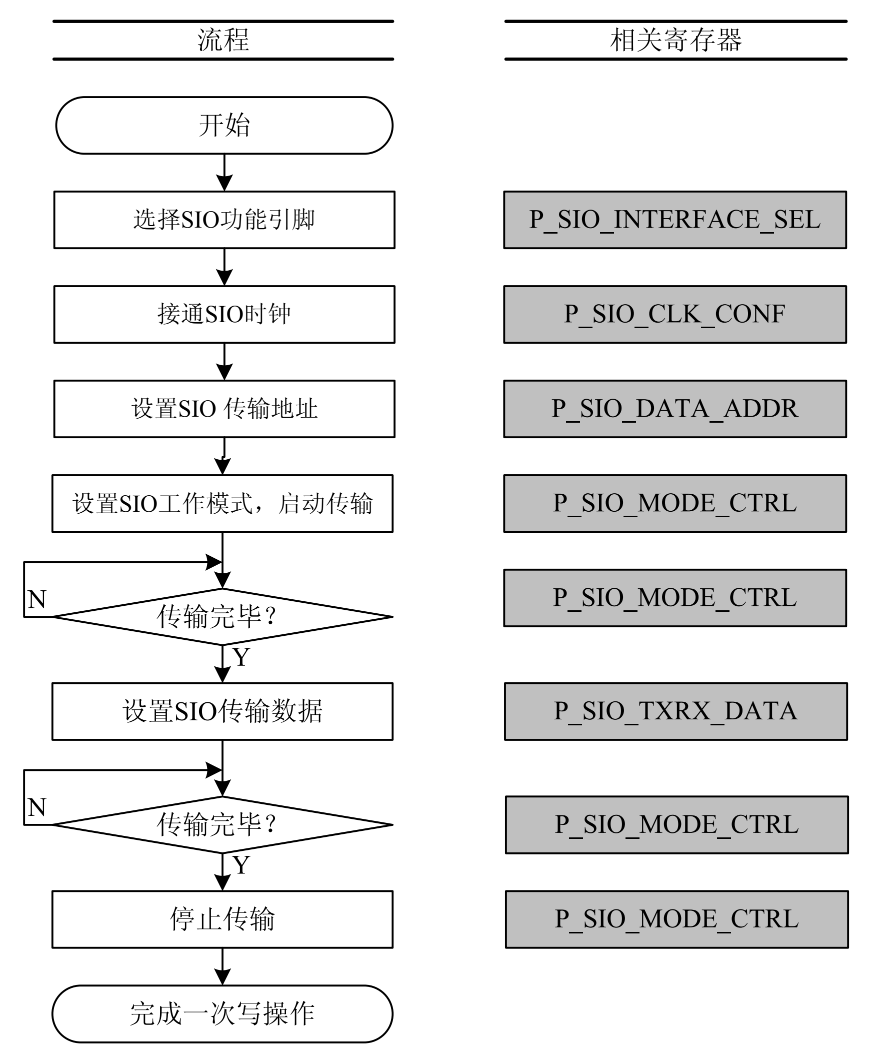


图 4.64 SIO 写操作流程图

1) 设置 P_SIO_INTERFACE_SEL 寄存器，选择管脚功能为 SIO 接口；
2) 设置 P_SIO_CLK_CONF 寄存器，使能 SIO 模块的时钟；
3) 设置 P_SIO_TXRX_ADDR 寄存器，选择外设的寻址地址（如果 SIO 设置了无地址模式，则可以跳过此步）；
4) 设置 P_SIO_MODE_CTRL 寄存器，根据需要设置 SIO 的工作模式，同时，使 SIO_START 位为 1，启动 SIO 总线的传输；
5) 反复查询 P_SIO_MODE_CTRL 的 b31 位 SIO_REQUEST 直至该位为 1，等待 SIO 传输就绪；
6) 向 P_SIO_TXRX_DATA 寄存器写入数据；
7) 反复查询 P_SIO_MODE_CTRL 的 b31 位 SIO_REQUEST 直至该位为 1，等待 SIO 传输就绪；
8) 传输完毕，设置 P_SIO_MODE_CTRL 寄存器的 SIO_START 位为 0。

参考代码如下：

```
#include “SPCE3200_Register.h”
#include “SPCE3200_Constant.h”
```

```
// 向 SPR4096 的 0x40 单元写入数据 0x55
int main(void)
{
    int i;
    *P_CLK_PLLAU_CONF |= C_PLLU_CLK_EN;                  // 使能 PLLU
    *P_SIO_INTERFACE_SEL |= C_SIO_PORT_SEL;              // 选择 SIO 接口
    *P_SIO_CLK_CONF = C_SIO_CLK_EN + C_SIO_RST_DIS;      // 使能 SIO 模块

    *P_SIO_DATA_ADDR = 0x40;                             // 设置操作地址
    *P_SIO_MODE_CTRL = C_SIO_DATA_8BIT                   // 选择 8 位数据宽度
                     | C_SIO_RWBIT_NORMAL                // 读写控制位选择普通模式
                     | C_SIO_CLK_27MDIV32                // 波特率选择 27MHz/32
                     | C_SIO_ADDR_24BIT                  // 选择 24 位地址宽度
                     | C_SIO_WRITE_MODE                  // 选择写操作方式
                     | C_SIO_TXRX_START;                 // 启动传输
    while((*P_SIO_MODE_CTRL & C_SIO_DATA_REQ) == 0);     // 等待传输结束
    *P_SIO_TXRX_DATA = 0x55;                             // 设置待写入的数据
    while((*P_SIO_MODE_CTRL & C_SIO_DATA_REQ) == 0);     // 等待传输结束
    *P_SIO_MODE_CTRL = C_SIO_TXRX_STOP;                  // 停止传输
    for(i = 0; i < 10000; i++);                          // 等待 SPR4096 内部烧写
    while(1);
    return 0;
}
```

使用 SIO 从外设读取数据的操作步骤如图 4.65 所示。

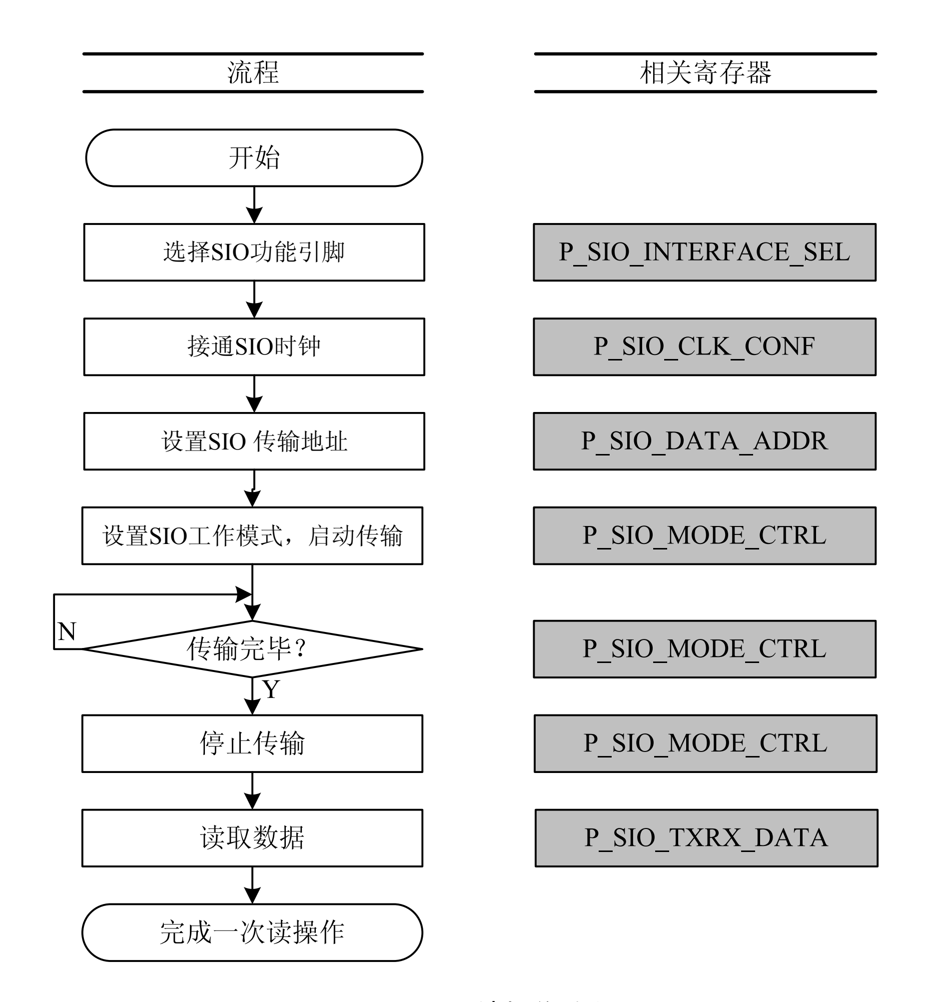


图 4.65 SIO 读操作流程图

设置 P_SIO_INTERFACE_SEL 寄存器，选择管脚功能为 SIO 接口；

1) 设置 P_SIO_CLK_CONF 寄存器，使能 SIO 模块的时钟；
2) 设置 P_SIO_TXRX_ADDR 寄存器，选择外设的寻址地址（如果 SIO 设置了无地址模式，则可以跳过此步）；
3) 设置 P_SIO_MODE_CTRL 寄存器，根据需要设置 SIO 的工作模式，同时，使 SIO_START 位为 1，启动 SIO 总线的传输；
4) 反复查询 P_SIO_MODE_CTRL 的 b31 位 SIO_REQUEST 直至该位为 1，等待 SIO 传输就绪；
5) 设置 P_SIO_MODE_CTRL 寄存器的 SIO_START 位为 0，停止传输；
6) 从 P_SIO_TXRX_DATA 寄存器读取数据；

参考代码如下：

```
#include “SPCE3200_Register.h”
#include “SPCE3200_Constant.h”
// 从 SPR4096 的 0x40 单元读取数据
int main(void)
{
```

```
        unsigned int Data;
        *P_CLK_PLLAU_CONF |= C_PLLU_CLK_EN;                      // 使能 PLLU
        *P_SIO_INTERFACE_SEL |= C_SIO_PORT_SEL;                  // 选择 SIO 接口
        *P_SIO_CLK_CONF = C_SIO_CLK_EN + C_SIO_RST_DIS;  // 使能 SIO 模块

        *P_SIO_DATA_ADDR = 0x40 ;                                // 设置操作地址
        *P_SIO_MODE_CTRL = C_SIO_DATA_8BIT                       // 选择 8 位数据宽度
                         | C_SIO_RWBIT_NORMAL                    // 读写控制位选择普通模式
                         | C_SIO_CLK_27MDIV32                    // 波特率选择 27MHz/32
                         | C_SIO_ADDR_24BIT                      // 选择 24 位地址宽度
                         | C_SIO_READ_MODE                       // 选择读操作方式
                         | C_SIO_TXRX_START;                     // 启动传输
        while((*P_SIO_MODE_CTRL & C_SIO_DATA_REQ) == 0);         // 等待传输结束
        *P_SIO_MODE_CTRL = C_SIO_TXRX_STOP;                      // 停止传输
        Data = *P_SIO_TXRX_DATA ;                                // 读取数据
        while(1);
        return 0;
    }
```

使用“自动传输”模式，但不使用 DS301_Ready 管脚：

在该模式下，SIO 模块可以自动查询 SPDS301 的状态寄存器，以确定 SPDS301 是否处于空闲状态，并自动重复发送过程，直至向 SPDS301 发送完毕用户设置的 AutoTx_Num 个数据。在该模式下不能接收数据。图 4.66 所示的是 SIO 模块运行于“自动传输”模式但不使用 DS301_Ready 管脚检测的运行流程。

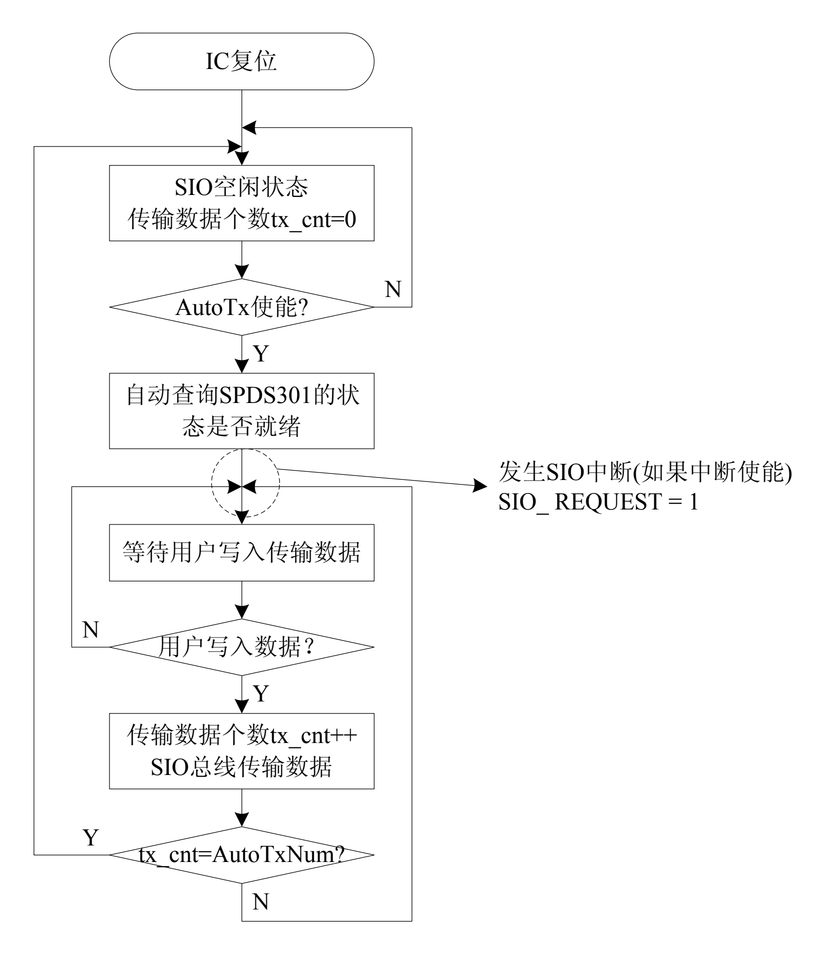


图 4.66 使用“自动传输”模式但不使用 DS301_Ready 管脚传输数据的操作流程

使用“自动传输”模式，并且使用 **DS301_Ready** 管脚：

在该模式下，SIO 模块查询 DS301_Ready 管脚的状态，以确定 SPDS301 是否处于空闲状态，并自动重复发送过程，直至向 SPDS301 发送完毕用户设置的 AutoTx_Num 个数据。在该模式下不能接收数据。图 4.67 所示的是 SIO 模块运行于“自动传输”模式并且使用 DS301_Ready 管脚检测的运行流程。

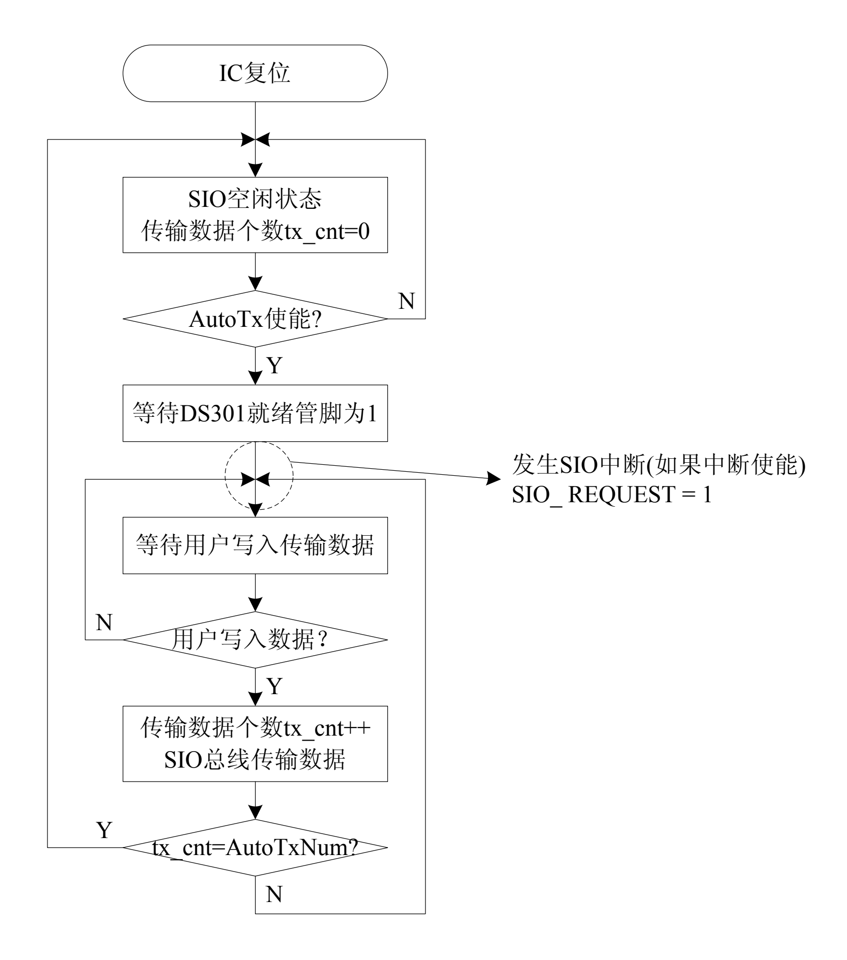


图 4.67 使用“自动传输”模式同时使用 DS301_Ready 管脚传输数据的操作流程

“自动传输”模式下进行数据传输的操作流程在这里不再详述。

4.11.7 注意事项

1，由于 SIO 控制器由 PLLU 提供工作频率，在使用 SIO 控制器之前，首先需要保证 P_CLK_PLLAU_CONF 寄存器的 b2 位为 1，使能 PLLU。

2，SIO 中断使能由 P_SIO_MODE_CTRL 的 b13 位控制，设置该位为 1 后将使能 SIO 中断，用户需要注意必须通过向该寄存器的 b7 位写入 1 来清除中断状态。关于 SIO 中断的正确操作如表 4.126 所示。

表 4.126 SIO 中断操作说明

SIO 工作模式	中断发生时 SIO 模块的状态	正确的操作
不使用“自动传输”模式	当 SIO 中断发生在读操作下，说明 SIO 接收到 1 个字节/字数据	读出 P_SIO_TXRX_DATA 寄存器内接收到的数据，然后向 P_SIO_MODE_CTRL 寄存器的 b7 位写入 1 清除中断标志，以便 SIO 可以继续接收数据。注意：即使清除了 SIO 的启动/停止传输控制位，同样需要清除中断标志

	当 SIO 中断发生在写操作下，说明 SIO 已经完成上一个写操作，等待下一个待写入的数据	向 P_SIO_TXRX_DATA 寄存器内写入数据后，当 SIO 模块完成数据传输后，将产生中断，此时，应向 P_SIO_MODE_CTRL 寄存器的 b7 位写入 1 以便清除中断状态。注意：即使清除了 SIO 的启动/停止传输控制位，同样需要清除中断标志
使用“自动传输”模式	SIO 中断产生说明 SIO 模块完成了上一次的数据传输，等待用户给出新的传输数据	向 P_SIO_TXRX_DATA 寄存器内写入数据后，当 SIO 模块完成数据传输后，将产生中断，此时，应向 P_SIO_MODE_CTRL 寄存器的 b7 位写入 1 以便清除中断标志

4.12 Nor 型 Flash 控制器

4.12.1 概述

闪速存储器（Flash Memory）具有非易失性，并且可轻易擦写。Flash 技术结合了 OTP 存储器的成本优势和 EEPROM 的可再编程性能，因此，得到越来越广泛的使用。

NOR 型 Flash 存储器是一种采用标准总线接口与处理器交互的 Flash ROM，其特点是高速随机字节存取能力，以及芯片内执行（XIP, eXecute In Place），这样应用程序可以直接在 flash 闪存内运行，不必再把代码读到系统 RAM 中。NOR flash 带有 SRAM 接口，有足够的地址引脚来寻址，可以很容易地存取其内部的每一个字节。

NOR 的传输效率很高，但是很低的写入和擦除速度大大影响了它的性能。在 1～4MB 小容量时具有很高的成本效益，更加安全，不容易出现数据故障，因此，主要应用以代码存储为主，多与运算相关。

对 Nor 型 Flash 的操作包括读操作、字写入操作、扇区/块/整片擦除操作和内部操作状态检测操作等。下面以 ST 公司的 SST39VF160 芯片为例，详细介绍以上几种操作。

SST39VF160 芯片具有 48 脚 TSOP 封装和 48 脚 TFBGA 封装两种形式。TSOP 封装的芯片引脚分布如图 4.68 所示。

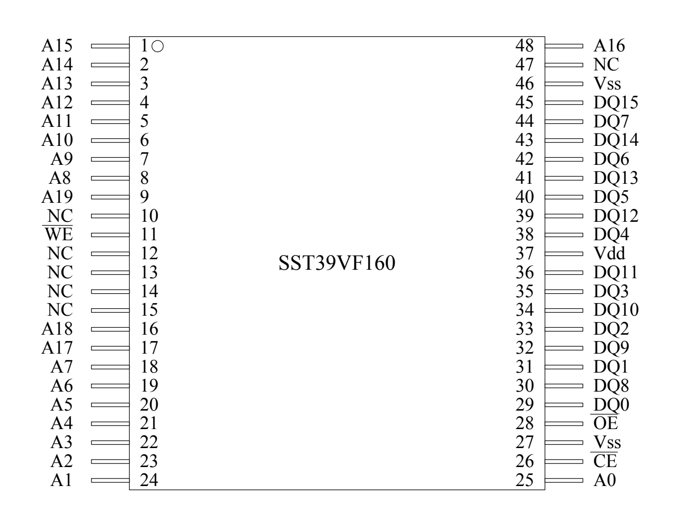


图 4.68 SST39VF160 引脚分布图

引脚描述如表 4.127 所示。

表 4.127 SST39VF160 引脚功能描述

引脚符号	引脚名称	功能描述
A19~A0	地址输入	提供存储器寻址地址。在扇区擦除中 A19～A11 地址线用来选择擦除哪一个扇区。在块擦除操作中 A19～A15 地址线用来选择擦除哪一个块
DQ15~DQ0	数据输入/输出	在读周期输出数据，在写周期接收写入的数据并内部锁存。在 $\overline{CE}$ 或 $\overline{OE}$ 为高电平时，数据线输入为高阻态
$\overline{CE}$	片选使能	芯片选择线，低电平有效
$\overline{OE}$	输出使能	数据输出使能线，低电平有效
$\overline{WE}$	写使能	写操作控制线
VDD	电源	为 SST39VF160 提供 2.7～3.6V 电源
VSS	地	
NC	不连接的引脚	

■ 读操作

SST39VF160 的读操作是由 $\overline{CE}$ 和 $\overline{OE}$ 信号线控制的。当两者都为低时，处理器就可以从 SST39VF160 的输出口读取数据。$\overline{CE}$ 是 SST39VF160 的片选线，当 $\overline{CE}$ 为高，芯片未被选中。$\overline{OE}$ 是输出使能信号线。当 $\overline{CE}$ 或 $\overline{OE}$ 中某一个为高时，SST39VF160 的数据线为高阻态。

SST39VF160 的读操作的时序如图 4.69 所示。

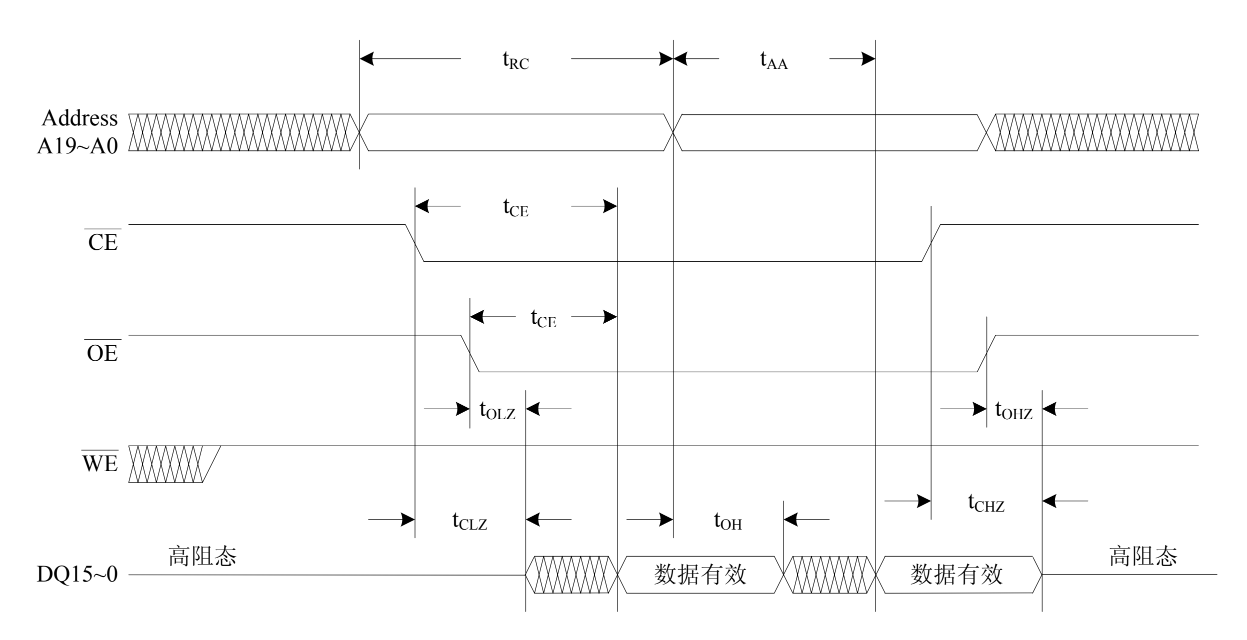


图 4.69 SST39VF160 的读时序

读时序的参数如表 4.128 所示。

表 4.128 读时序参数表（单位：ns）

符号	描述	SST39VF160-70		SST39VF160-90	
		最小值	最大值	最小值	最大值
t_{RC}	整个读周期时间	70		90	
t_{CE}	读周期中芯片使能时间		70		90
t_{AA}	地址操作时间		70		90
t_{OE}	输出数据使能时间		35		45
t_{CLZ}	从 $\overline{CE}$ 变低到动态输出时间	0		0	
t_{OLZ}	从 $\overline{OE}$ 变低到动态输出时间	0		0	
t_{CHZ}	从 $\overline{CE}$ 变高到数据线高阻态时间		20		30
t_{OHZ}	从 $\overline{OE}$ 变高到数据线高阻态时间		20		30
t_{OH}	地址改变时输出数据保持时间	0		0	

■ 字写入操作

SST39VF160 的写操作主要是以一个字接一个字的方式进行写入的。在写入之前，扇区中如果



有数据（非 0xFF），则必须首先进行充分地擦除。写操作分三步进行：

第一步，送出“软件数据保护”的 3 字节；

第二步，送出地址和数据；

第三步，内部写入处理阶段，这个阶段在第 4 个 $\overline{WE}$ 或 $\overline{CE}$ 的上升沿时被初始化。被初始化后，内部写入处理就将在 20μs 时间内完成。在内部写入阶段，任何指令都将被忽略。字写入操作流程如图 4.70 所示。

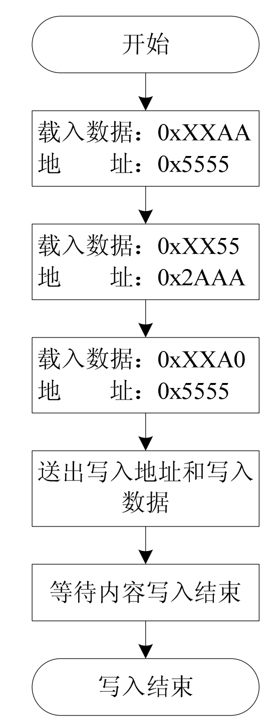


图 4.70 SST39VF160 的字写入流程图

SST39VF160 的字写入时序如

图 4.71 和图 4.72 所示。

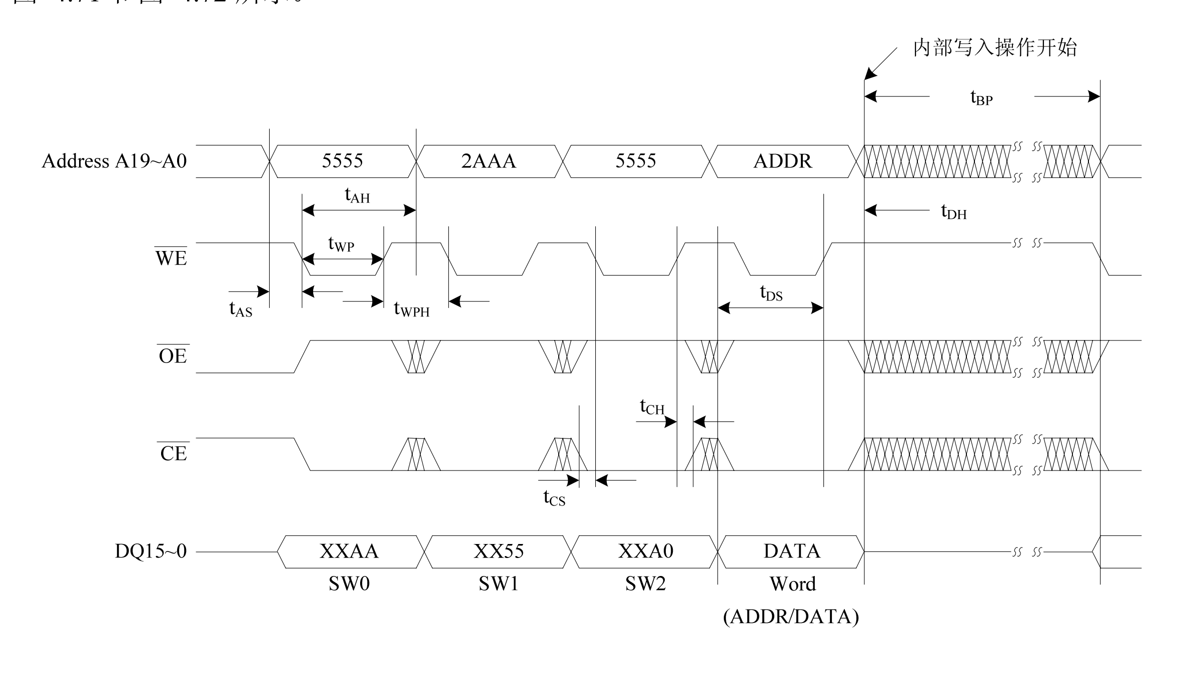


图 4.71 通过 $\overline{WE}$ 控制的字写入时序图

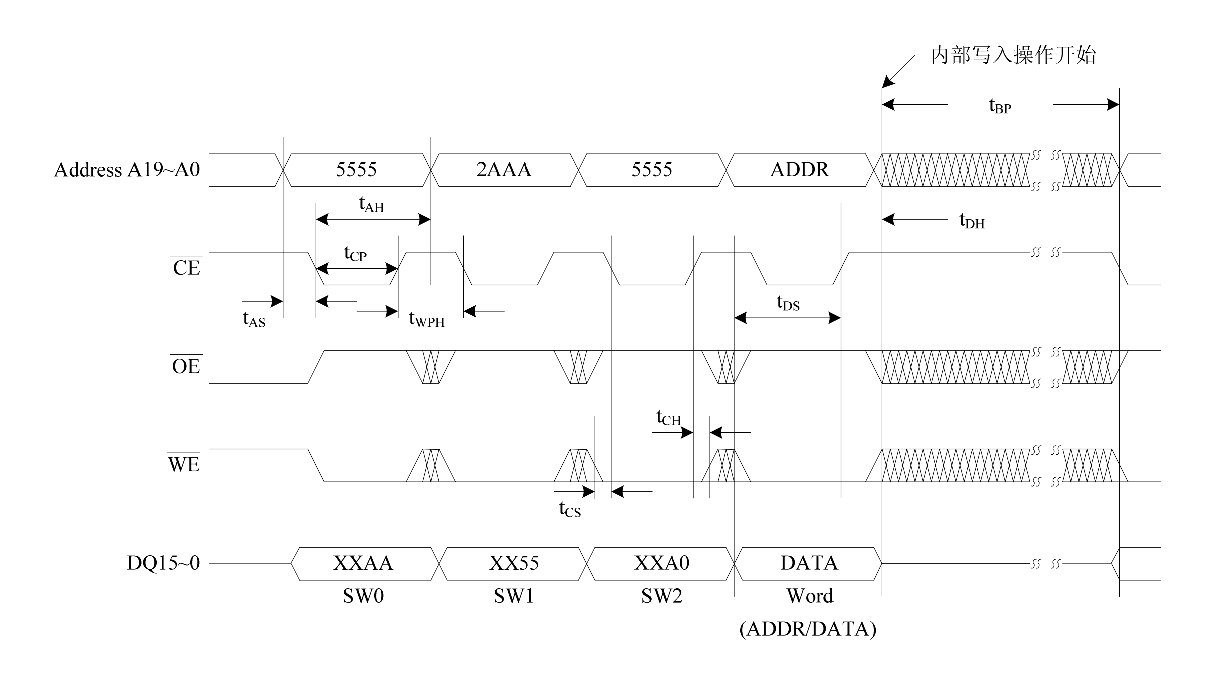


图 4.72 通过 $\overline{CE}$ 控制的字写入时序图

SST39VF160 的写入及擦除时序参数表如表 4.129 所示。

表 4.129 写入及擦除时序参数表

符号	描述	最小值	最大值	单位
t_{BP}	字写入时间		20	μs
t_{AS}	地址建立时间	0		ns
t_{AH}	地址保持时间	30		ns
t_{CS}	$\overline{WE}$ 和 $\overline{CE}$ 建立时间	0		ns
t_{CH}	$\overline{WE}$ 和 $\overline{CE}$ 保持时间	0		ns
t_{OES}	$\overline{OE}$ 高电平建立时间	0		ns
t_{OEH}	$\overline{OE}$ 高电平保持时间	10		ns
t_{CP}	$\overline{CE}$ 脉宽	40		ns
t_{WP}	$\overline{WE}$ 脉宽	40		ns



符号	描述	最小值	最大值	单位
t_{WPH}	$\overline{WE}$ 高电平脉宽	30		ns
t_{CPH}	$\overline{CE}$ 高电平脉宽	30		ns
t_{DS}	数据建立时间	30		ns
t_{DH}	数据保持时间	0		ns
t_{IDA}	软件 ID 操作和退出时间		150	ns
t_{SE}	扇区擦除		25	ms
t_{BE}	块擦除		25	ms
t_{SCE}	片擦除		100	ms

■ 扇区/块/整片擦除操作

扇区或块擦除操作允许 SST39VF 160 以一个扇区接一个扇区，或一个块接一个块地进行擦除。扇区是统一的 2K Word（16 位）大小。块是统一的 32K Word 大小。扇区操作通过执行 6 字节的指令序列来进行，这个指令序列中包括扇区擦除指令（0x30）和扇区地址（SectorAddress）。块操作也通过 6 字节的指令序列来进行，这个指令序列中包括块擦除指令（0x50）和块地址（BlockAddress）。扇区或块的地址在 $\overline{WE}$ 的第 6 个下降沿处锁存，指令字节（0x30 或 0x50）在 $\overline{WE}$ 的第 6 个上升沿处锁存。之后，开始内部擦除操作。除了可以使用内部操作状态检测方法判断内部擦除是否结束之外，在内部擦除阶段其他的指令都将被忽略。

SST39VF160 还提供一个整片擦除的功能，允许使用者一次性快速擦除整个存储器（存储阵列的每个单位都为 1）。整片擦除同样通过执行 6 字节的指令序列来进行，6 字节指令序列中包括片擦除指令（0x10）和字节序列最后的地址 0x5555。图 4.73 所示为擦除操作的流程图。

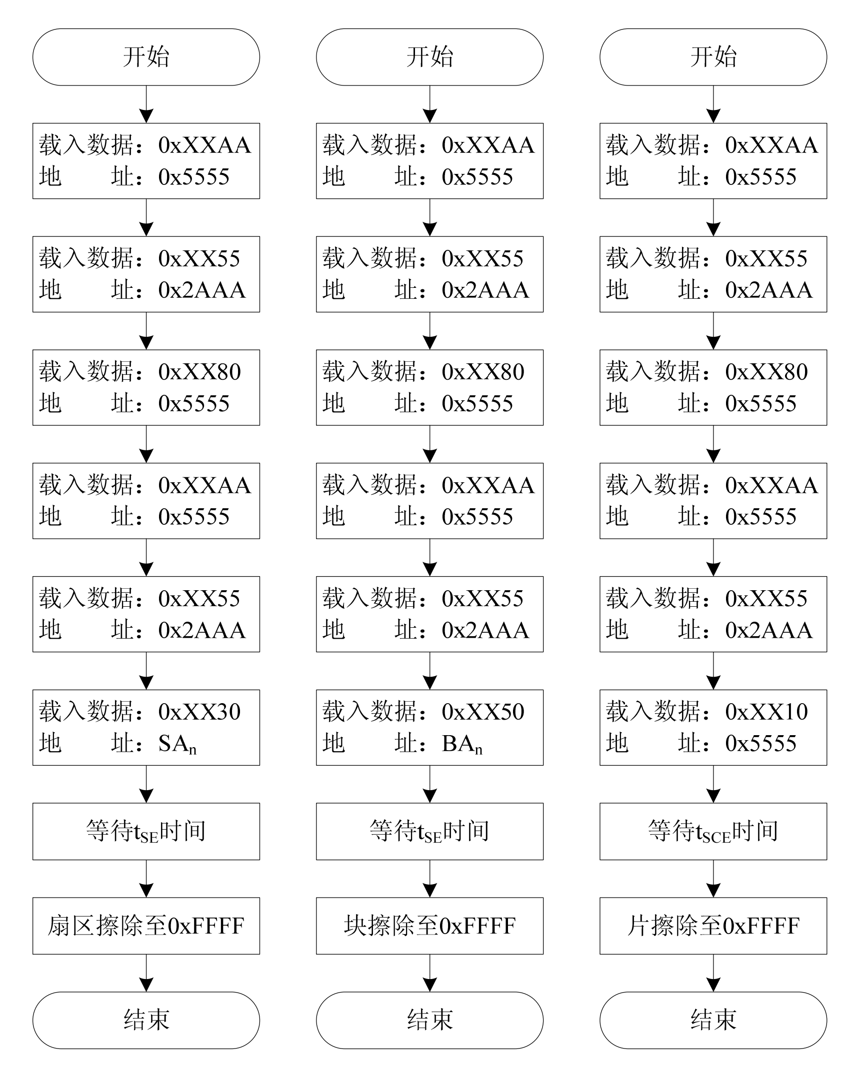


图 4.73 SST39VF160 的擦除操作流程

图 4.74 所示为块擦除时序图，该器件支持 $\overline{CE}$ 信号控制的块擦除操作。BA_n 为块地址。

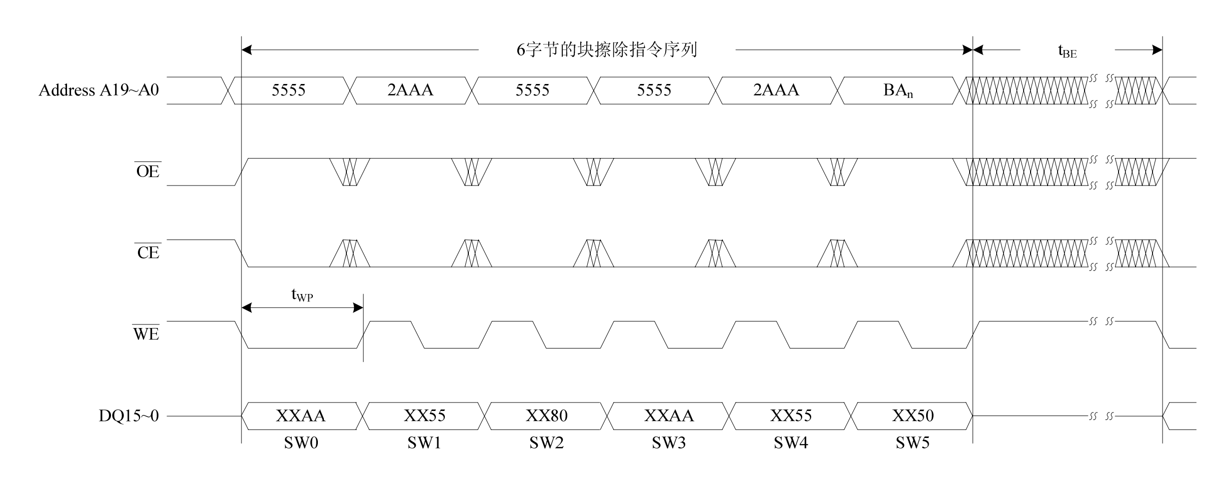


图 4.74 SST39VF160 的块擦除时序图

图 4.75 所示为扇区擦除时序图，该器件支持 $\overline{CE}$ 信号控制的扇区擦除操作。SA_n 为块地址。

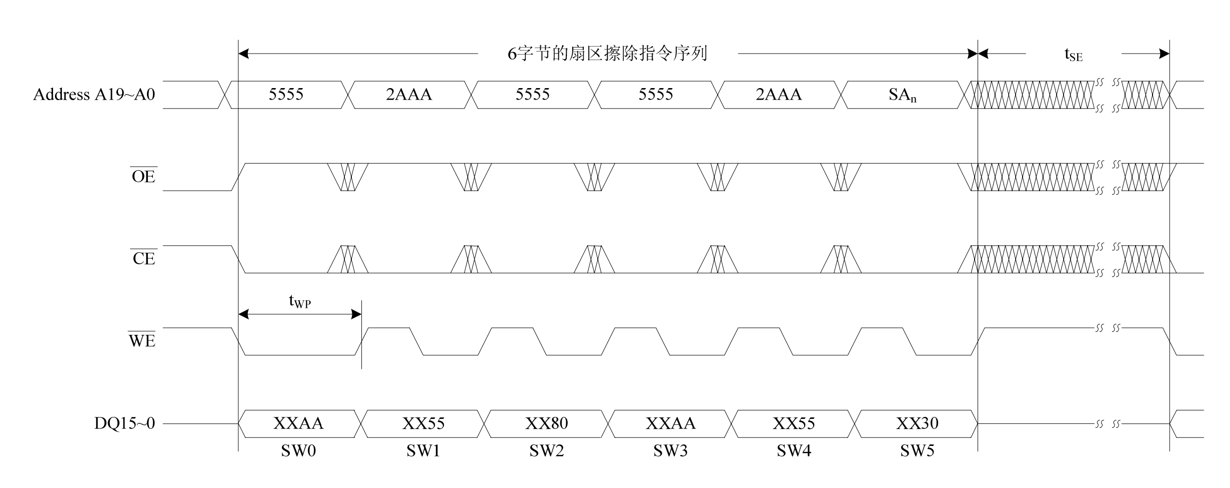


图 4.75 SST39VF160 的扇区擦除时序图

图 4.76 所示为整片擦除时序图，该器件支持 $\overline{CE}$ 信号控制的整片擦除操作。

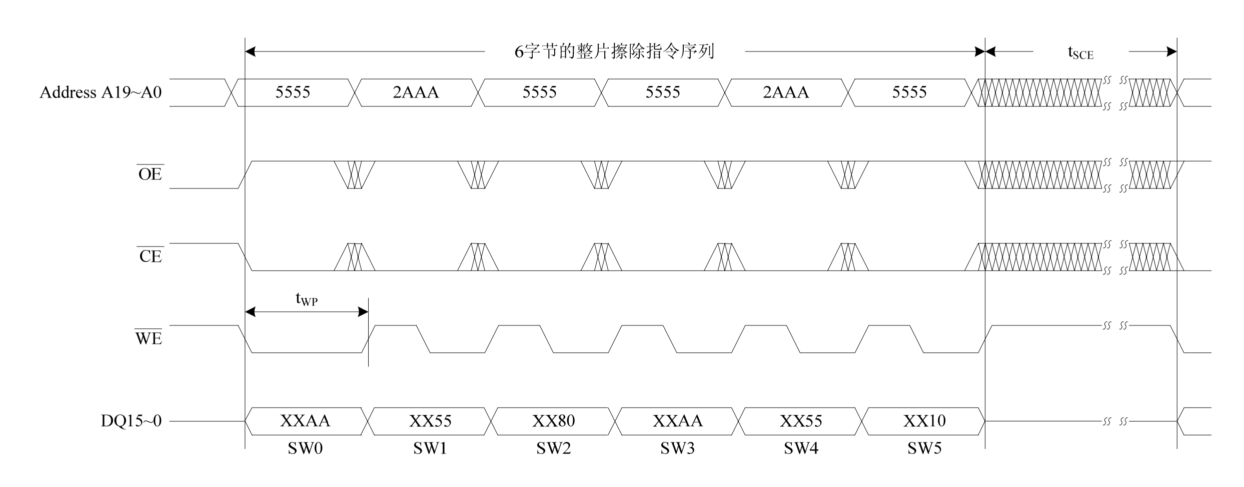


图 4.76 SST39VF160 的整片擦除时序图

■ 内部操作状态检测

SST39VF160 提供两种软件方式来检测内部操作是否完成。软件检测方式涉及两个状态位：$\overline{Data}$ Polling Bit（DQ7）和 Toggle bit（DQ6）。在送完写命令序列或擦除命令序列之后的 $\overline{WE}$ 的上升沿，写入结束检测功能被使能，同时内部写入或擦除操作也被初始化，此后可以通过写入结束检测操作查询内部操作是否完成。这里，只介绍 Toggle bit 方式的检测操作。

在内部写入或擦除的过程中，任何对 DQ6 连续的读操作，芯片都会产生一个不断翻转的 1 和 0 输出。当内部写入或擦除完成时，DQ6 将停止翻转。其时序图和操作流程图分别如图 4.77 和图 4.78 所示。

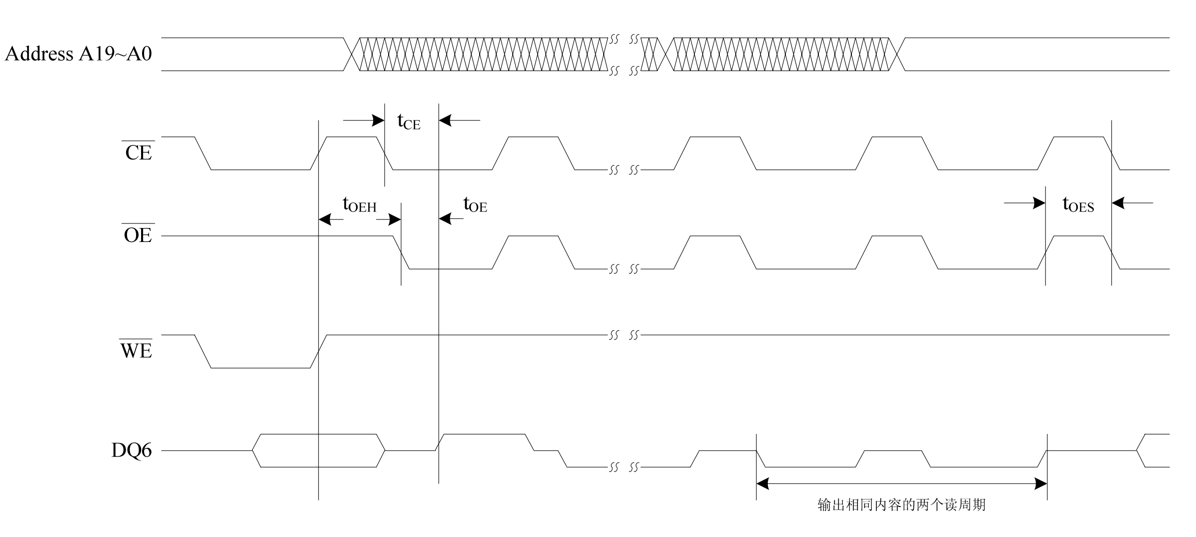


图 4.77 Toggle bit 时序图

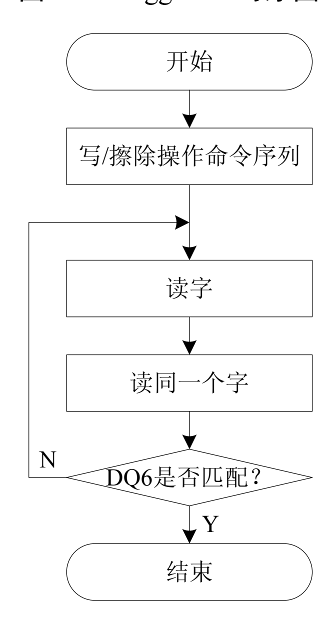


图 4.78 Toggle Bit 方式检测操作流程图

SPCE3200 内部内置 Nor 型 Flash 控制器，可以通过 DRAM 接口实现对外部 Nor 型 Flash 或 ROM 的访问，从而实现 boot 程序或应用程序的存储。为满足 ROM 或 Flash 的要求，可以对访问速度进行编程。

4.12.2 特性

SPCE3200 内部的 Nor 型 Flash 接口隶属于 Memory Interface Unit 部件管辖，具有以下特点：

- 总线接口与 DRAM 和 SRAM 复用
- 数据总线支持 16bit 宽度
- 可在线对 Flash 进行擦写

4.12.3 引脚描述

SPCE3200 的 Nor 型 Flash 的接口与 DRAM 公用总线。其引脚描述如表 4.130 所示。

表 4.130 Nor 型 Flash 接口引脚对应表



引脚名称	引脚号	引脚属性	引脚功能	复用功能
CE	15	O	Flash 片选	ROMCSN
OE	221	O	输出使能	DRAM_CAS
WE	220	O	写使能	DRAM_WE
DQ0	218	I/O	数据总线	DRAM_D0
DQ1	217	I/O		DRAM_D1
DQ2	216	I/O		DRAM_D2
DQ3	215	I/O		DRAM_D3
DQ4	214	I/O		DRAM_D4
DQ5	212	I/O		DRAM_D5
DQ6	211	I/O		DRAM_D6
DQ7	210	I/O		DRAM_D7
DQ8	209	I/O		DRAM_D16
DQ9	208	I/O		DRAM_D17
DQ10	207	I/O		DRAM_D18
DQ11	205	I/O		DRAM_D19
DQ12	204	I/O		DRAM_D20
DQ13	203	I/O		DRAM_D21
DQ14	202	I/O		DRAM_D22
DQ15	201	I/O		DRAM_D23
A0	256	O	地址总线	DRAM_A0
A1	1	O		DRAM_A1
A2	2	O		DRAM_A2
A3	3	O		DRAM_A3
A4	247	O		DRAM_A4
A5	246	O		DRAM_A5
A6	245	O		DRAM_A6
A7	243	O		DRAM_A7
A8	242	O		DRAM_A8
A9	241	O		DRAM_A9

引脚名称	引脚号	引脚属性	引脚功能	复用功能
A10	255	O		DRAM_A10
A11	239	O		DRAM_A11
A12	14	O		DRAM_A12
A13	224	O		DRAM_BA0
A14	254	O		DRAM_BA1
A15	251	O		DRAM_DQM0
A16	249	O		DRAM_DQM1
A17	250	O		DRAM_DQM2
A18	248	O		DRAM_DQM3
A19	238	O		DRAM_D8
A20	237	O		DRAM_D9
A21	232	O		DRAM_D10
A22	231	O		DRAM_D11
A23	230	O		DRAM_D12

SPCE3200 外接的 Nor 型 Flash 的虚拟地址被分配在 0x9E000000～0x9EFFFFFF 范围内。在使用时，可以直接访问该地址范围内的存储单元。

4.12.4 寄存器描述

SPCE3200 内部的 Nor 型 Flash 控制器共有 2 个控制寄存器，如表 4.131 所示。这两个寄存器主要控制着对 Nor 型 Flash 的擦除或写操作。

表 4.131 Nor 型 Flash 控制器相关寄存器列表

寄存器名称	助记符	地址
NOR 命令控制寄存器	P_NOR_COMMAND_CTRL	0x880700BC
NOR 块设置寄存器	P_NOR_BANK_SETUP	0x880700C4
DRAM 接口选择寄存器	P_DRAM_INTERFACE_SEL	0x88200008
DRAM GPIO 设置寄存器	P_DRAM_GPIO_SETUP	0x88200050
DRAM GPIO 输入数据寄存器	P_DRAM_GPIO_INPUT	0x88200070

如果使用 DRAM 接口（DRAM 接口与 NOR 接口复用）接口复用为 GPIO 功能，可以对下面的寄存器操作。寄存器的各位对应得引脚关系如表 4.132：

表 4.132 DRAM 接口复用为 GPIO 引脚与寄存器位对应关系



引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
ROMCSN	15	bit[0]	不可以
DRAM_A11	239	bit[1]	不可以
DRAM_A12	14	bit[2]	不可以
DRAM_BA1	254	bit[3]	不可以

DRAM 接口选择寄存器：P_DRAM_INTERFACE_SEL(0x88200008)

SPCE3200 的 NOR 型 FLASH 接口引脚与 DRAM 接口引脚复用，而且其中部分引脚可以作为 GPIO 使用，在使用时需要设置 DRAM 接口选择寄存器以便使其工作在需要的接口模式。

表 4.133 P_DRAM_INTERFACE_SEL(0x88200008)

位	b31~b25	b24	b23~b17	b16	b15~b9	b8	b7~b0
读/写	-	R/W	-	R/W	-	R/W	-
默认值	-	0	-	0	-	0	-
名称	-	SW_DRAM_BA1	-	SW_DRAM_A12	-	SW_DRAM_A11	-

SW_DRAM_BA1 b24 DRAM_BA1 引脚功能选择位：
0：作为 GPIO 使用
1：作为 DRAM BA1 块选择引脚使用

SW_DRAM_A12 b16 DRAM_A12 引脚功能选择位：
0：作为 GPIO 使用
1：作为 DRAM A12 地址选择引脚使用

SW_DRAM_A11 b8 DRAM_A11 引脚功能选择位：
0：作为 GPIO 使用
1：作为 DRAM A11 地址选择引脚使用

DRAM GPIO 设置寄存器：P_DRAM_GPIO_SETUP(0x88200050)

DRAM GPIO 设置寄存器设置 DRAM 接口复用为 GPIO 时的引脚输出使能、输出数据、上拉电阻输入、下拉电阻输入。

表 4.134 P_DRAM_GPIO_SETUP(0x88200050)

位	b27~b24	b23~b20	b19~b16	b15~b12	b11~b8	b7~b4	b3~b0
读/写	R/W	-	R/W	-	R/W	-	R/W
默认值	0	-	0	-	0	-	0
名称	DRAM_PD	-	DRAM_PU	-	DRAM_OE	-	DRAM_O

DRAM_PD　　b27~b24　　DRAM 接口作为 GPIO 使用时的下拉电阻输入使能位：
　　0: 不使能下拉电阻输入
　　1: 使用下拉电阻输入

DRAM_PU　　b19~b16　　DRAM 接口作为 GPIO 使用时的上拉电阻输入使能位：
　　0：使能上拉电阻输入
　　1：不使能上拉电阻输入

DRAM_OE　　b11~b8　　DRAM 接口作为 GPIO 使用时的输出使能位：
　　0：不输出使能
　　1：输出使能

DRAM_O　　b3~b0　　DRAM 接口作为 GPIO 使用时的输出数据

DRAM GPIO 输入数据寄存器：P_DRAM_GPIO_INPUT(0x88200070)

DRAM GPIO 输入寄存器保存 DRAM 接口复用为 GPIO 时的外部输入数据。

表 4.135 P_DRAM_GPIO_INPUT(0x88200070)

位	b31~b20	b19~b16	b15~b0
读/写	-	R/W	-
默认值	-	0	-
名称	-	DRAM_INPUT	-

DRAM_INPUT　　b19~b16　　DRAM 接口作为 GPIO 使用时的输入数据

NOR 命令控制寄存器：P_NOR_COMMAND_CTRL(0x880700BC)

NOR 命令控制寄存器，可以打开或关闭对 Nor 型 Flash 的命令寄存器的操作。

表 4.136 P_NOR_COMMAND_CTRL(0x880700BC)

位	b31~b1	b0
读/写	-	W
默认值	-	0
名称	-	COMMAND_EN

COMMAND_EN　　b0　　NOR 型 FLASH 命令写使能位：
　　0：普通模式，该模式下向 NOR 型 FLASH 写入的数据均为普通数据
　　1：命令模式，该模式下向 NOR 型 FLASH 写入的数据均为命

令数据

NOR 块设置寄存器：P_NOR_BANK_SETUP(0x880700C4)

SPCE3200 内的 NOR 型 FLASH 控制器在执行擦除或者编程操作时，需要以 2MB 大小的块来进行。在进行写或擦除之前，需将块号写入该寄存器。

表 4.137 P_NOR_BANK_SETUP(0x880700C4)

位	b31~b8	b7~b0
读/写	-	W
默认值	-	0
名称	-	BANK_ADDR

BANK_ADDR	b7~b0	用户需要以 2MB 为单位，计算当前写入地址或擦除地址属于哪个块，并将块号写入该寄存器

4.12.5 基本操作

读操作

对 NOR 型 FLASH 的读操作比较简单，直接访问 NOR 型 FLASH 所在的地址空间的地址即可。例如：欲读取 0x9E000000 地址单元的数据，则可使用如下代码进行：

```
int *addr;
int temp;
addr = 0x9e000000;
temp = *addr;
```

擦除操作

对 NOR 型 FLASH 的擦除操作需要遵循芯片本身规定的命令序列来进行。同时，需要设置 P_NOR_COMMAND_CTRL 寄存器和 P_NOR_BANK_SETUP 寄存器，以便控制器可以准确对擦除区域操作。

以 NOR 型 FLASH 的扇区擦除操作为例，说明擦除操作的典型步骤。图 4.79 所示的使用 SPCE3200 开发板对 NOR 型 FLASH 进行扇区擦除的编程步骤。

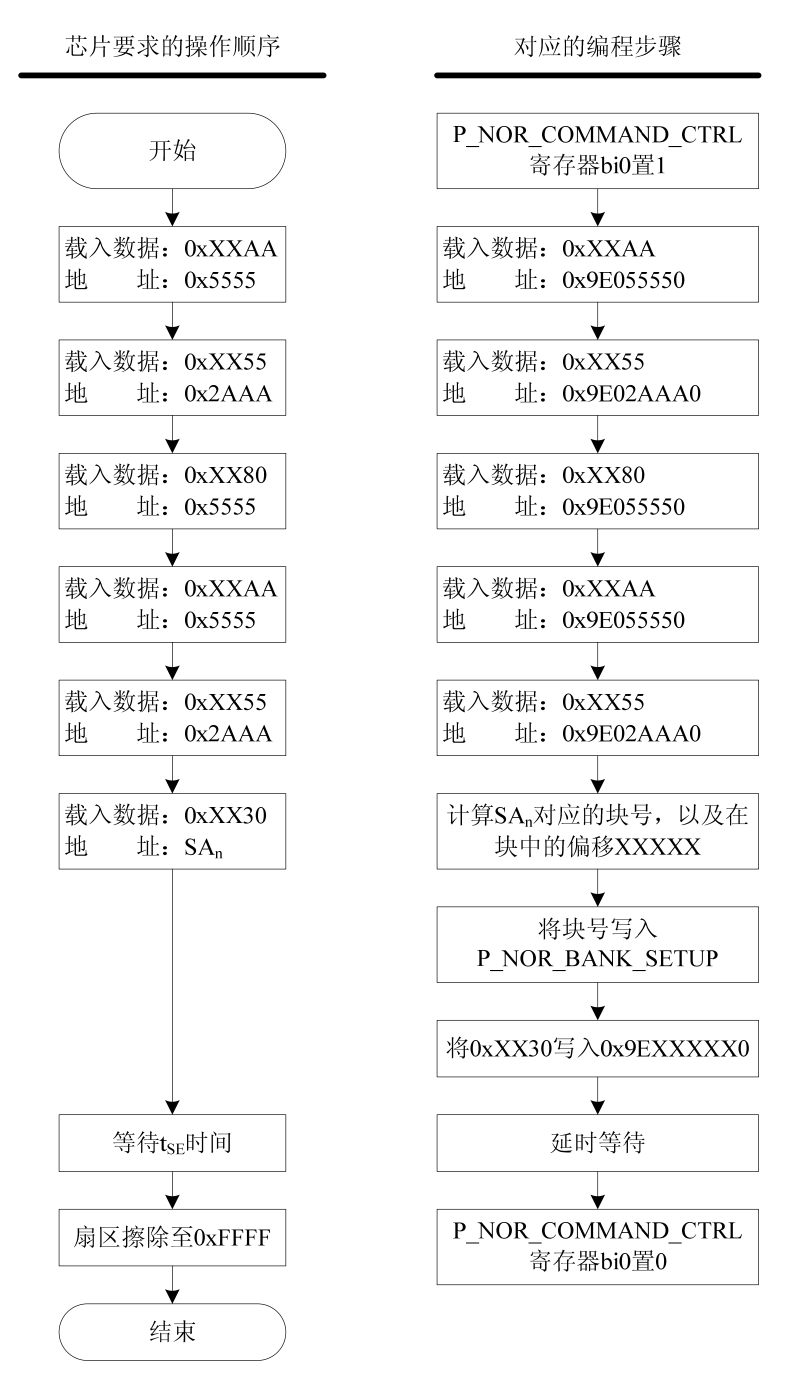


图 4.79 扇区擦除操作步骤

参考代码如下：

```
#define P_NOR_COMMAND_CTRL   (volatile unsigned int *)0x880700bc
#define P_NOR_BANK_SETUP     (volatile unsigned int *)0x880700c4

#define C_Flash_BaseAddr     0x9e000000
#define C_CMD1_Addr          (volatile unsigned int *)0x9e055550
#define C_CMD2_Addr          (volatile unsigned int *)0x9e02aaa0
```

```
#define C_CMD3_Addr         (volatile unsigned int *)0x9e055550
#define C_CMD4_Addr         (volatile unsigned int *)0x9e055550
#define C_CMD5_Addr         (volatile unsigned int *)0x9e02aaa0
#define C_CMD6_Addr         (volatile unsigned int *)0x9e055550

#define C_CMD_Data1         0xaa            // 芯片要求的命令
#define C_CMD_Data2         0x55            // 芯片要求的命令
#define C_CMD_Data3         0x80            // 芯片要求的命令
#define C_CMD_Data4         0xaa            // 芯片要求的命令
#define C_CMD_Data5         0x55            // 芯片要求的命令
#define C_SECTOR_ERASE      0x30            // 擦除命令
#define C_SECTOR_ERASE_TIME    500000       // 擦除扇区延时时间
#define C_SECTOR_SIZE       2048            // 2048 字/扇区，1 word = 16 bits

extern void NorFlash_SectorErase(unsigned int addr);
//==========================================================================
// 语法格式:      void NorFlash_SectorErase(unsigned int addr)
// 实现功能:      扇区擦除(4KB/Sector)
// 参数:          addr -    擦除地址(低 12bit 将被忽略)
// 返回值:    无
//==========================================================================
void NorFlash_SectorErase(unsigned int addr)
{
    addr -= C_Flash_BaseAddr;                   // 去掉基地址 0x9e000000
    addr = addr / 2;                            // 字地址转换成字节地址
    addr = addr & (~(C_SECTOR_SIZE - 1));       // 将低 12bit 置为 0
    addr = NorFlash_SetAddress(addr);           // 以 2MB 为单位，计算属于哪个块，并将该
// 块写入 NOR 块设置寄存器
    addr = C_Flash_BaseAddr + (addr << 4);      // 根据硬件连接，转换操作地址
    *P_NOR_COMMAND_CTRL = 1;                    // 作为命令写入数据

    *C_CMD1_Addr = C_CMD_Data1;                 // 根据芯片要求，写入命令 1，0xAA
    *C_CMD2_Addr = C_CMD_Data2;                 // 根据芯片要求，写入命令 2，0x55
    *C_CMD3_Addr = C_CMD_Data3;                 // 根据芯片要求，写入命令 3，0x80
    *C_CMD4_Addr = C_CMD_Data4;                 // 根据芯片要求，写入命令 4，0xAA
    *C_CMD5_Addr = C_CMD_Data5;                 // 根据芯片要求，写入命令 5，0x55
```

```
        *(unsigned int *)addr = C_SECTOR_ERASE;     // 擦除扇区地址

        delay(C_SECTOR_ERASE_TIME);                 // 延时芯片规定时间
        *P_NOR_COMMAND_CTRL = 0;                    // 切换回普通模式
}

//==============================================================================
// 语法格式：      unsigned int NorFlash_SetAddress(unsigned int addr)
// 实现功能：      为控制器设置操作地址(内部调用)
// 参数：          addr -    原始操作地址
// 返回值：    控制器操作地址
//==============================================================================
unsigned int NorFlash_SetAddress(unsigned int addr)
{
        unsigned i = 0;
        while(addr > 0x0fffff)                      // 如果地址大于 2MB
        {
                addr -= 0x100000;                   // 减去 2MB 地址
                i++;                                // 块号加 1
        }
        *P_NOR_BANK_SETUP = i;                      // 设置块号
        return(addr);
}

//==============================================================================
// 语法格式：      void delay(unsigned int clk)
// 实现功能：      延时一段时间(内部调用)
// 参数：          clk  -    延时时间控制
// 返回值：    无
//==============================================================================
void delay(unsigned int clk)
{
        unsigned int i;
        for(i = 0; i < clk; i++);
}
```

编程操作（写操作）

对 NOR 型 FLASH 的写操作与擦除操作类似，按照 NOR 型 FLASH 芯片规定的命令顺序操作即可，这里不再细述操作流程。

参考代码如下：

```
#define P_NOR_COMMAND_CTRL   (volatile unsigned int *)0x880700bc
#define P_NOR_BANK_SETUP     (volatile unsigned int *)0x880700c4

#define C_Flash_BaseAddr     0x9e000000
#define C_CMD1_Addr          (volatile unsigned int *)0x9e055550
#define C_CMD2_Addr          (volatile unsigned int *)0x9e02aaa0
#define C_CMD3_Addr          (volatile unsigned int *)0x9e055550
#define C_CMD4_Addr          (volatile unsigned int *)0x9e055550
#define C_CMD5_Addr          (volatile unsigned int *)0x9e02aaa0
#define C_CMD6_Addr          (volatile unsigned int *)0x9e055550

#define C_CMD_Data1          0xaa            // 芯片要求的命令
#define C_CMD_Data2          0x55            // 芯片要求的命令
#define C_CMD_Data3          0x80            // 芯片要求的命令
#define C_CMD_Data4          0xaa            // 芯片要求的命令
#define C_CMD_Data5          0x55            // 芯片要求的命令
#define C_WORD_WRITE         0xa0            // 写入命令

#define C_WRITE_DELAY_TIME   500             // 写入延时时间
#define C_SECTOR_SIZE        2048            // 2048 字/扇区，1 word = 16 bits

//==========================================================================
// 语法格式：    void NorFlash_Write32(unsigned int addr, unsigned int data)
// 实现功能：    写入 32bit 数据
// 参数：        addr -   操作地址
//               data -   写入数据
// 返回值：      无
//==========================================================================
void NorFlash_Write32(unsigned int addr, unsigned int data)
{
    unsigned int uiTemp;
```

```
        if(addr >= C_Flash_BaseAddr)                    // 是否大于基地址
        {
            addr -= C_Flash_BaseAddr;                   // 减掉基地址
            uiTemp = addr;                              // 临时保存写入地址
            //write low 16 bits
            addr >>= 1;                                 // 转换成芯片字节地址，字的低地址
            addr = NorFlash_SetAddress(addr);           // 设置 NOR 块设置寄存器
            addr = C_Flash_BaseAddr + (addr << 4);      // 根据硬件连线转换为实际操作地址
            NorFlash_WordWrite(addr, data & 0x0000ffff);// 写入数据低 16 位
            //write high 16 bits
            addr = uiTemp;                              // 恢复地址
            addr >>= 1;                                 // 转换成芯片字节地址
            addr++;                                     // 字的高地址
            addr = NorFlash_SetAddress(addr);           // 设置 NOR 块设置寄存器
            addr = C_Flash_BaseAddr + (addr << 4);      // 根据硬件连线转换为实际操作地址
            NorFlash_WordWrite(addr, (data >> 16) & 0x0000ffff);// 写入数据高 16 位
        }
}

//=========================================================================
// 语法格式：    void NorFlash_WordWrite(unsigned int addr, unsigned data)
// 实现功能：    Nor Flash 写入操作流程(内部调用)
// 参数：        addr -    操作地址
//               data -    写入数据
// 返回值：      无
//=========================================================================
void NorFlash_WordWrite(unsigned int addr, unsigned data)
{
        *P_NOR_COMMAND_CTRL = 1;                        // 作为命令写入数据
        data &= 0x0000ffff;
        *C_CMD1_Addr = C_CMD_Data1;                     // 根据芯片要求，写入命令 1，0xAA
        *C_CMD2_Addr = C_CMD_Data2;                     // 根据芯片要求，写入命令 2，0x55
        *C_CMD3_Addr = C_WORD_WRITE;                    // 根据芯片要求，写入命令 3，0xA0
        *(unsigned int *)addr = data;                   // 写入数据
        delay(C_WRITE_DELAY_TIME);                      // 延时 20us
        *P_NOR_COMMAND_CTRL = 0;                        // 切换回普通模式
```

```
}

//==============================================================================
// 语法格式：      unsigned int NorFlash_SetAddress(unsigned int addr)
// 实现功能：      为控制器设置操作地址(内部调用)
// 参数：          addr -      原始操作地址
// 返回值：    控制器操作地址
//==============================================================================
unsigned int NorFlash_SetAddress(unsigned int addr)
{
    unsigned i = 0;
    while(addr > 0x0fffff)                      // 如果地址大于 2MB
    {
        addr -= 0x100000;                       // 减去 2MB 地址
        i++;                                    // 块号加 1
    }
    *P_NOR_BANK_SETUP = i;                      // 设置块号
    return(addr);
}

//==============================================================================
// 语法格式：      void delay(unsigned int clk)
// 实现功能：      延时一段时间(内部调用)
// 参数：          clk   -     延时时间控制
// 返回值：    无
//==============================================================================
void delay(unsigned int clk)
{
    unsigned int i;
    for(i = 0; i < clk; i++);
}
```

4.13 Nand 型 Flash 控制器

4.13.1 概述

对于许多消费类音视频产品而言，Nand 型 Flash 存储设备是一种比硬盘驱动器更好的存储方案，这在不超过 4GB 的低容量应用中表现得犹为明显。随着人们持续追求功耗更低、重量更轻和性能更

佳的产品，NAND 正被证明极具吸引力。

Nand 型 Flash 结构能提供极高的单元密度，可以达到高存储密度，并且写入和擦除的速度也很快。NAND 器件使用复杂的 I/O 口来串行地存取数据，对于 16 位的器件，并行闪存大约需要 40 多个 I/O 引脚，相对而言，NAND 器件仅需 24 个引脚。

Nand 型 Flash 使用 8 个引脚用来传送控制、地址和数据信息，同时配合 ALE、CLE、CEN、WEN、REN 等控制管脚，实现 Flash 的读写管理。

典型的 Nand 型 Flash 器件的引脚分布如图 4.80 所示。

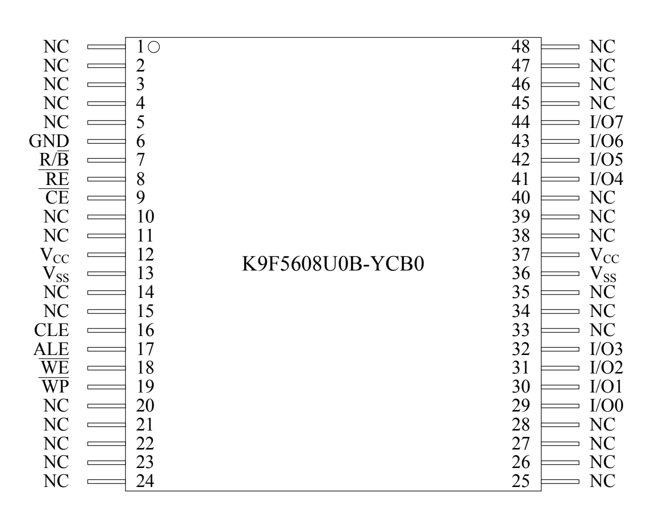


图 4.80 K9F5608U0b 引脚分布图

引脚描述如表 4.138 所示。

表 4.138 Nand 型 Flash 引脚功能描述

引脚符号	引脚名称	功能描述
I/O7～I/O0	数据输入/输出	I/O 口是用来输入指令、地址和数据，并在读周期时输出数据的。当芯片未被选中或输出禁能时，I/O 口呈高阻态。
CLE	命令锁存控制	CLE 引脚上的输入电平用来控制打开/关闭指令送入指令寄存器的通路。当 CLE 为高时，I/O 口在 $\overline{WE}$ 信号的上升沿将指令锁存至指令寄存器。
ALE	地址锁存控制	ALE 引脚上的输入电平用来控制打开/关闭地址送入地址寄存器的通路。当 ALE 为高时，I/O 口在 $\overline{WE}$ 信号的上升沿将地址锁存至地址寄存器。
$\overline{CE}$	芯片使能	低电平使能的片选控制线。 当芯片处于忙状态时，该引脚即使变高也将被忽略。
$\overline{RE}$	读使能	该引脚为串行数据输出控制线。当它为低电平时，内部数据将输出至 I/O 端口。输出数据在该引脚下降后 t_{REA} 时间内有效，同时，内部列地址计数器将加 1。

引脚符号	引脚名称	功能描述
$\overline{WE}$	写使能	该引脚对 I/O 端口的写入进行控制。指令、地址和数据都会在该引脚的上升沿被锁存。
$\overline{WP}$	写保护	该引脚提供在电源波动情况下，对器件不可预料的写入或擦除的保护。当该引脚为低电平时，内部高电压发生器被复位。
$R/\overline{B}$	读/忙输出	该引脚的输出说明了器件目前的操作状态。当它为低电平时，表明某个写入、擦除或随机读操作正在进行，当这个操作完成，该引脚才会重新回到高电平状态。它是一个漏极开路输出，而且在芯片未选中或输出未使能时，不会进入高阻态。
V_{CC}	电源	器件的供电电源。
V_{SS}	地	
NC	不连接的引脚	
GND	GND 输入使能备用区	在进行连续行读操作时，当要读取包括备用区在内的数据时，要把 GND 接 V_{SS} 或拉低；如果不要读取包括备用区在内的数据时，则将该引脚接 V_{CC} 或置高。

Nand 型 Flash 器件的存储空间的组织结构由若干个块（Block）组成，每个块又由 32 或 64 个页（Page）组成，每个页则包含了 528 或 2112 个字节（列）。其中，具有 528 字节的页容量的器件，每个块由 32 个页组成；具有 2112 字节的页容量的器件，每个块由 64 个页组成。

在 528 字节或 2112 字节的页中，分别包含了 16 字节或 64 字节的备用区域，该区域与其他的 512 字节或 2048 字节没有区别，但是，一般用备用区域来做坏块标识或损耗均衡等工作。

Nand 型 Flash 器件的存储空间的组织结构如图 4.81 所示。

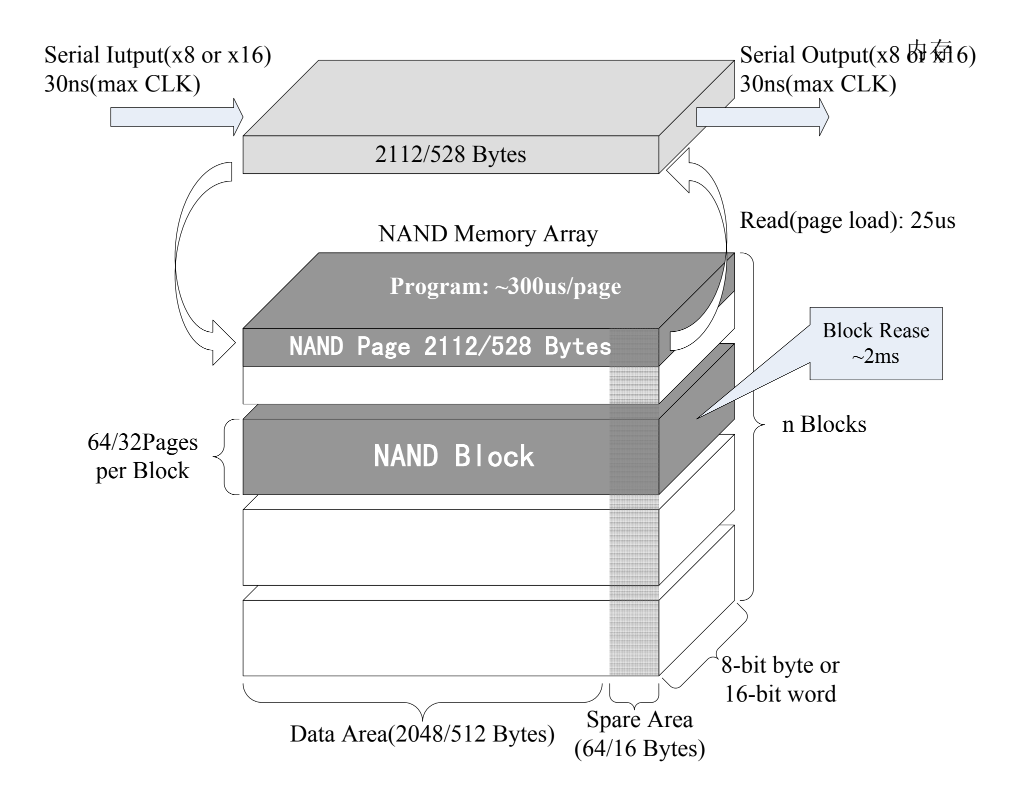


图 4.81 Nand 型 Flash 存储空间的组织结构

在对 Nand 型 Flash 进行读或写操作时，都以页为单位进行；擦除操作则以块为单位进行。

对 Nand 型 Flash 器件内部存储单元的寻址地址通过 8 位端口传送，这样，可以有效的节省了引脚的数量，并能保持不同密度器件引脚的一致性，允许系统可以在电路不做改动的情况下升级为高容量的存储器件。

Nand 型 Flash 器件通过 CLE 和 ALE 信号线实现 I/O 口上指令和地址的复用。指令、地址和数据都通过拉低 $\overline{WE}$ 和 $\overline{CE}$ 从 I/O 口写入器件中。CLE 和 ALE 是否为高决定了当前写入的数据是命令还是地址，或者是数据。一些命令只需要一个总线周期完成，例如，复位命令、读命令和读状态命令等；另外一些命令，例如页写入和块擦除命令等，则需要 2 个周期，即在 CLE 有效的情况下启动两次 $\overline{WE}$ 信号来写入命令。表 4.139 列举出了以 K9F5608U08 为例的 Nand 型 Flash 具备的命令。

表 4.139 K9F5608U0B 具备的指令和功能

功能	第一个周期	第二个周期	器件忙时是否接收
读方式 1	0x00/0x01	-	否
读方式 2	0x50	-	否
读芯片 ID 号	0x90	-	否
复位	0xff	-	是
页写入	0x80	0x10	否

功能	第一个周期	第二个周期	器件忙时是否接收
回拷贝	0x00	0x8a	否
块擦除	0x60	0xd0	否
读当前状态	0x70	-	是

SPCE3200 内嵌了 NAND 型 Flash 控制器，以便可以方便的连接 Nand 型 Flash 存储单元，提供大容量存储解决方案。该控制器提供标准的带有 ECC 纠错码计算电路的 8 位 Nand 型 Flash 总线接口，同时，还支持连接智能媒体卡。

4.13.2 特性

SPCE3200 内部的 Nand 型 Flash 控制器具有以下特性：

- 完全可编程的 CLE 和 ALE 时序，支持多种 Nand 型 Flash
- 提供查询/中断/DMA 的访问方式
- 支持 528 字节~2112 字节的页容量
- 可编程的读/写时钟周期
- 可编程的 ALE 时钟周期

4.13.3 引脚描述

SPCE3200 的 Nand 型 Flash 的接口引脚与 SPI 和 SD 卡接口复用，在使用时需要设置相关的寄存器以便使其工作在 Nand 型 Flash 接口模式。Nand 型 Flash 接口的引脚描述如表 4.140 所示。

表 4.140 Nand 型 Flash 接口引脚对应表

引脚名称	引脚号	引脚属性	引脚功能
NF_ALE	127	O	地址锁存使能，高电平有效
NF_WPN	126	O	写保护使能，低电平有效
NF_CLE	125	O	命令锁存使能，高电平有效
NF_REN	124	O	读数据使能，低电平有效
NF_WEN	123	O	写数据使能，低电平有效
NF_CEN	119	O	芯片片选，低电平有效
NF_RDY	118	I	Nand 型 Flash 就绪输入，高电平有效
NF_D0	117	I/O	数据总线
NF_D1	116	I/O	
NF_D2	115	I/O	
NF_D3	112	I/O	
NF_D4	111	I/O	

NF_D5	110	I/O	
NF_D6	109	I/O	
NF_D7	108	I/O	

4.13.4 结构

Nand 型 Flash 控制器的结构如图 4.82 所示。

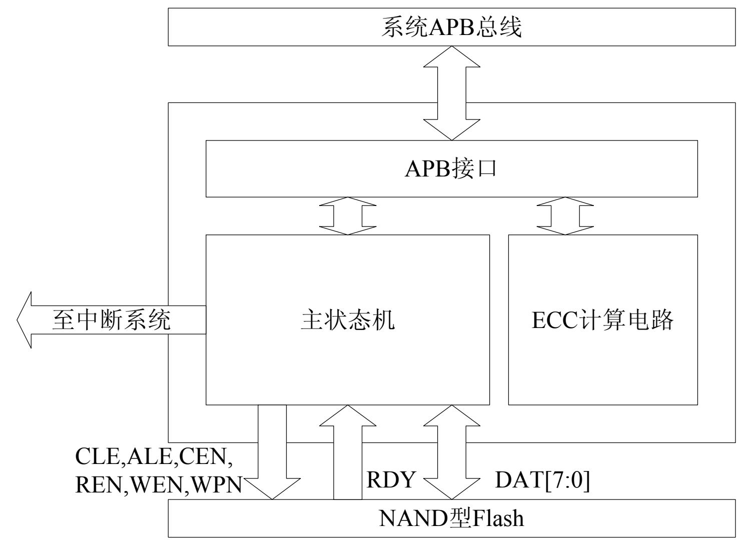


图 4.82 Nand 型 Flash 控制器结构

4.13.5 寄存器描述

Nand 型 Flash 控制器共有 19 个寄存器，如表 4.141 所示。通过对这 19 个寄存器的操作，即可通过 Nand 型 Flash 接口扩展 Nand 型 Flash 存储设备。下面将对这些寄存器一一进行说明。

表 4.141 Nand 型 Flash 控制器相关寄存器列表

寄存器名称	助记符	地址
NAND 接口选择寄存器	P_NAND_INTERFACE_SEL	0x882000A4
NAND GPIO 设置寄存器	P_NAND_GPIO_SETUP	0x8820002C
NAND GPIO 上下拉寄存器	P_NAND_GPIO_PULL	0x88200030
NAND GPIO 输入数据寄存器	P_NAND_GPIO_INPUT	0x8820006C
NAND GPIO 外部中断寄存器	P_NAND_GPIO_INT	0x88200088
NAND 时钟配置寄存器	P_NAND_CLK_CONF	0x882100A8
NAND 中断控制寄存器	P_NAND_INT_CTRL	0x88190014
NAND 控制寄存器	P_NAND_MODE_CTRL	0x88190000
NAND 命令寄存器	P_NAND_CLE_COMMAND	0x88190004
NAND 地址寄存器	P_NAND_ALE_ADDR	0x88190008



NAND 发送数据寄存器	P_NAND_TX_DATA	0x8819000C
NAND 接收数据寄存器	P_NAND_RX_DATA	0x88190010
NAND 中断状态寄存器	P_NAND_INT_STATUS	0x88190018
NAND 真实行校验码寄存器	P_NAND_ECC_TRUELP	0x8819001C
NAND 真实列校验码寄存器	P_NAND_ECC_TRUECP	0x88190020
NAND 对比行校验码寄存器	P_NAND_ECC_CMPLP	0x88190024
NAND 对比列校验码寄存器	P_NAND_ECC_CMPCP	0x88190028
NAND 校验状态寄存器	P_NAND_ECC_STATUS	0x8819002C
NAND 校验控制寄存器	P_NAND_ECC_CTRL	0x88190030

如果使用 Nand 型 Flash 接口复用为 GPIO 功能，可以对下面的寄存器操作。寄存器的各位对应引脚关系如表 4.142：

表 4.142 Nand 接口复用为 GPIO 引脚与寄存器位对应关系

引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
NF_ALE	127	bit[0]	可以
NF_WPN	126	bit[1]	可以
NF_CLE	125	bit[2]	可以
NF_REN	124	bit[3]	可以
NF_WEN	123	bit[4]	不可以
NF_CEN	119	bit[5]	不可以
NF_RDY	118	bit[6]	不可以
NF_D0	117	bit[7]	不可以
NF_D1	116	bit[8]	不可以
NF_D2	115	bit[9]	不可以
NF_D3	112	bit[10]	不可以
NF_D4	111	bit[11]	不可以
NF_D5	110	bit[12]	不可以
NF_D6	109	bit[13]	不可以
NF_D7	108	bit[14]	不可以
SPI_CSN	128	bit[15]	不可以

NAND GPIO 设置寄存器：P_NAND_GPIO_SETUP(0x8820002C)

NAND GPIO 设置寄存器设置 NAND 接口复用为 GPIO 时的引脚输出使能和输出数据。

表 4.143 P_NAND_GPIO_SETUP(0x8820002C)

位	b31~b16	b15~b0
读/写	R/W	R/W
默认值	0	0
名称	NFLASH_GPIO_OE	NFLASH_GPIO_O

NFLASH_GPIO_OE　b31-b16　　NFLASH 接口作为 GPIO 使用时的输出使能位：

0：不使能输出

1：使能输出

NFLASH_GPIO_O　　b15-b0　　NFLASH 接口作为 GPIO 使用时的输出数据

NAND GPIO 上/下拉寄存器：P_NAND_GPIO_PULL(0x88200030)

NAND GPIO 上/下拉寄存器设置 NAND 接口复用为 GPIO 时的上拉电阻输入使能和下拉电阻输入使能功能。

表 4.144 P_GPIO_NFLASH_PULL（0x88200030）

位	b31~b16	b15~b0
读/写	R/W	R/W
默认值	1	1
名称	NFLASH_GPIO_PD	NFLASH_GPIO_PU

NFLASH_GPIO_PD　b31~b16　　NFlash 接口作为 GPIO 使用时的下拉电阻输入使能位：

0：不使能下拉电阻输入

1：使能下拉电阻输入

NFLASH_GPIO_PU　b15~b0　　NFlash 接口作为 GPIO 使用时的上拉电阻输入使能位：

0：使能上拉电阻输入

1：不使能上拉电阻输入

NAND GPIO 输入数据寄存器：P_NAND_GPIO_INPUT(0x8820006C)

NAND GPIO 输入寄存器保存 NAND 接口复用为 GPIO 时的外部输入数据。

表 4.145 P_NAND_GPIO_INPUT(0x8820006C)

位	b15~b0
读/写	R

默认值	0
名称	NFLASH_INPUT

NFLASH_INPUT　　b15-b0　　NFlash 接口作为 GPIO 使用时的输入数据

NAND GPIO 外部中断寄存器：P_NAND_GPIO_INT(0x88200088)

NAND GPIO 外部中断寄存器可进行中断使能、中断触发沿设置、中断标志清除等操作。

表 4.146 P_NAND_GPIO_INT(0x88200088)

位	b27~b24	b23~b20	b19~b16	b15~b12	b11~b8	b7~b4	b3~b0
读/写	R/W	-	R/W	-	R/W	-	R/W
默认值	0	-	0	-	0	-	0
名称	NFLASH_FI	-	NFLASH _RI	-	NFLASH _FIEN	-	NFLASH _RIEN

NFLASH_FI　　b27-b24　　NFlash GPIO 下降沿中断标志位：

读 0：没有发生下降沿中断

读 1：发生下降沿中断

写 0：无意义

写 1：清除中断标志

NFLASH _RI　　b19-b16　　NFlash GPIO 上升沿中断标志位：

读 0：没有发生上升沿中断

读 1：发生上升沿中断

写 0：无意义

写 1：清除中断标志

NFLASH _FIEN　　b11-b8　　NFlash GPIO 下降沿中断使能位：

0：不使能下降沿中断

1：使能下降沿中断

NFLASH _RIEN　　b3-b0　　NFlash GPIO 上升沿中断使能位：

0：不使能上升沿中断

1：使能上升沿中断

作为 NAND FLASH 控制器需要设置以下寄存器：

NAND 接口选择寄存器：P_NAND_INTERFACE_SEL(0x882000A4)

SPCE3200 的 Nand 型 Flash 的接口引脚与 SPI 和 SD 卡接口复用，在使用时需要设置 NAND 接

口选择寄存器以便使其工作在 Nand 型 Flash 接口模式。

表 4.147 P_NAND_INTERFACE_SEL(0x882000A4)

位	b31~b1	b0
读/写	-	R/W
默认值	-	0
名称	-	NFLASH_EN

NFLASH_EN　　b0　　Nand 型 Flash 接口使能位：

0：不使能为 NAND FLASH 接口

1：使能为 NAND FLASH 接口

NAND 时钟配置寄存器：P_NAND_CLK_CONF(0x882100A8)

NAND 时钟配置寄存器用于使能或禁止 Nand 型 Flash 控制器模块时钟，或软复位 Nand 型 Flash 控制器模块。

表 4.148 P_NAND_CLK_CONF(0x882100A8)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	FLASH_RST	FLASH_STOP

FLASH_RST　　b1　　NAND FLASH 模块时钟复位位：

0：NAND FLASH 模块时钟复位

1：NAND FLASH 模块时钟不复位

FLASH_STOP　　b0　　NAND FLASH 模块时钟使能位：

0：NAND FLASH 模块时钟停止

1：NAND FLASH 模块时钟使能

NAND 中断控制寄存器：P_NAND_INT_CTRL(0x88190014)

NAND 中断控制寄存器用于使能或禁止 Nand 控制器的中断。Nand 型 Flash 控制器使用 IRQ41 中断。

表 4.149 P_NAND_INT_CTRL(0x88190014)

位	b31~b6	b5	b4	b3	b2	b1	b0
读/写		R/W	R/W	R/W	R/W	R/W	R/W
默认值		0	0	0	0	0	0

名称		INTEN5	INTEN4	INTEN3	INTEN2	INTEN1	INTEN0

INTEN5	b5	命令丢失（CMDLOSS）中断使能位： 0：禁止 1：使能 当 Flash 控制器不忙的时候向该寄存器写入数据时，Flash 控制器将启动 CLE 信号输出，并将写入 NAND 命令寄存器的命令送至八位数据总线，同时会启动 WEN 信号的输出，以便将命令写入 Nand 型 Flash。如果当控制器忙的时候向该寄存器写入数据，将引起命令丢失（CMDLOSS）标志位置 1，同时触发命令丢失中断
INTEN4	b4	读数据错误（RDMISS）中断使能位： 0：禁止 1：使能 当读缓冲区满时，超过一定时间未读取数据，则发生读数据错误中断
INTEN3	b3	写数据丢失（WRLOSS）中断使能位： 0：禁止 1：使能 当写缓冲区空时，超过一定时间未写入数据，则发生写数据丢失中断
INTEN2	b2	RDY 信号中断使能位： 0：禁止 1：使能 当 RDY 信号有上升沿输入时触发该中断
INTEN1	b1	读缓冲区满中断使能位： 0：禁止 1：使能 当读缓冲区满时，产生该中断
INTEN0	b0	写缓冲区空中断使能位： 0：禁止 1：使能 当写缓冲区空时，产生该中断

NAND 控制寄存器：P_NAND_MODE_CTRL(0x88190000)

NAND 控制寄存器用于对 Nand 型 Flash 存储器访问时序的设定。

表 4.150 P_NAND_MODE_CTRL(0x88190000)

位	b31-b28	b27-b16	b15-b14	b13-b12	b11-b10	b9-b8
读/写	-	R/W	R/W	R/W	R/W	R/W
默认值	-	0	0	0	0	0
名称	-	TBYTES	RDHIGH	RDLOW	WRHIGH	WRLOW
位	b7-b6	b5-b4	b3	b2	b1	b0
读/写	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	1	1	0
名称	ALECYC	RDTYPE	WRCMD	FLWPN	FLCEN	FLEN

TBYTES　　b27-b16　　本次传输的数据量：

如果本次需要传输的数据为 N，在设置的时候写 N - 1

RDHIGH　　b15-b14　　REN（读操作使能）高电平持续时钟周期数量：

00：1 个时钟周期

01：2 个时钟周期

10：3 个时钟周期

11：4 个时钟周期

这里的时钟周期来自于 27MHz 晶振

RDLOW　　b13-b12　　REN（读操作使能）低电平持续时钟周期数量：

00：1 个时钟周期

01：2 个时钟周期

10：3 个时钟周期

11：4 个时钟周期

这里的时钟周期来自于 27MHz 晶振

WRHIGH　　b11-b10　　WEN（写操作使能）高电平持续时钟周期数量：

00：1 个时钟周期

01：2 个时钟周期

10：3 个时钟周期

11：4 个时钟周期

这里的时钟周期来自于 27MHz 晶振

WRLOW　　b9-b8　　WEN（写操作使能）低电平持续时钟周期数量：

00：1 个时钟周期

01：2 个时钟周期

10：3 个时钟周期

		11：4 个时钟周期 这里的时钟周期来自于 27MHz 晶振
ALECYC	b7-b6	当 P_NAND_ALE_ADDR 单元被写入时触发多少个写信号： 00：1 个时钟周期 01：2 个时钟周期 10：3 个时钟周期 11：4 个时钟周期 该寄存器主要用来设置地址周期数量。即，在 ALE 有效期间地址被分为多少个字节被写入
RDTYPE	b5-b4	设置读操作的启动时刻： 00：在 RDY 信号上升沿之后读取 01：在 CLE 下降沿之后读取 10：在 ALE 下降沿之后读取 11：保留
WRCMD	b3	读/写命令控制位： 0：读命令 1：写命令
FLWPN	b2	WPN 信号线控制位： 0：WPN 输出低电平 1：WPN 输出高电平
FLCEN	b1	CEN 信号线控制位： 0：CEN 输出低电平 1：CEN 输出高电平
FLEN	b0	Flash 接口使能位： 0：Flash 接口禁止 1：Flash 接口使能 如需连接 Flash 存储器，该位必须设置为 1

NAND 命令寄存器：P_NAND_CLE_COMMAND(0x88190004)

当 Flash 控制器不忙的时候向该寄存器写入数据时，Flash 控制器将启动 CLE 信号输出，并将写入 NAND 命令寄存器的命令送至八位数据总线，同时会启动 WEN 信号的输出，以便将命令写入 Nand 型 Flash。如果当控制器忙的时候向该寄存器写入数据，将引起命令丢失（CMDLOSS）标志位置 1，同时触发命令丢失中断。

表 4.151 P_NAND_CLE_COMMAND(0x88190004)

位	b7~b0

读/写	R/W
默认值	0
名称	FL_CLE

FL_CLE　　b7-b0　　Flash 控制命令：
向该寄存器写入数据将触发 CLE 信号以及 WEN 信号的输出

NAND 地址寄存器：P_NAND_ALE_ADDR(0x88190008)

当 Flash 控制器不忙的时候向该寄存器写入数据时，Flash 控制器将启动 ALE 信号输出，并将写入该寄存器的地址根据 P_NAND_MODE_CTRL 寄存器的 ALECYC 设置从最低字节开始依次写入数据总线，同时启动 WEN 信号的输出，以便写入 Nand 型 Flash。

表 4.152 P_NAND_ALE_ADDR(0x88190008)

位	**b31~b0**
读/写	R/W
默认值	0
名称	FL_ALE

FL_ALE　　b31-b0　　Flash 操作地址：
向该寄存器写入数据将触发 ALE 信号以及 WEN 信号的输出。WEN 信号的输出数量由 P_NAND_MODE_CTRL 寄存器的 ALECYC 确定

NAND 发送数据寄存器：P_NAND_TX_DATA(0x8819000C)

当 P_NAND_INT_STATUS 寄存器的 WREMPTY 位为 1 的时候必须向该寄存器写入数据，否则，将引起写数据丢失（WRLOSS）位将被置 1，同时触发写数据丢失中断，此时，Nand 型 Flash 会认为控制器已经完成写操作，从而拉低自己的 RDY 引脚，开始编程操作，从而导致用户此后无法写入数据。

表 4.153 P_NAND_TX_DATA(0x8819000C)

位	**b31~b0**
读/写	R/W
默认值	0
名称	FL_WD

FL_WD　　b31-b0　　向 NAND Flash 写入的数据



NAND 接收数据寄存器：P_NAND_RX_DATA(0x88190010)

当 P_NAND_INT_STATUS 寄存器的 RDFULL 位为 1 的时候必须从该寄存器读出数据，否则，将引起读数据错误（RDMISS）位将被置 1，同时触发读数据错误中断，此时，控制器停止 REN 信号的输出，导致 Nand 型 Flash 认为读取操作结束，从而使此后读出操作无效。

表 4.154 P_NAND_RX_DATA(0x88190010)

位	b31~b0
读/写	R/W
默认值	0
名称	FL_RD

FL_RD　　b31-b0　　从 NAND Flash 读出的数据

NAND 中断状态寄存器：P_NAND_INT_STATUS(0x88190018)

NAND 中断状态寄存器用于查询控制器的中断状态。

表 4.155 P_NAND_INT_STATUS(0x88190018)

位	b7	b6	b5	b4	b3	b2	b1	b0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
默认值	0	0	0	0	0	0	0	0
名称	RDY	BUSY	CMDLOSS	RDMISS	WRLOSS	RDYRISE	RDFULL	WREMPTY

RDY　　b7　　Nand Flash 就绪标志位：

0：Nand Flash 忙

1：Nand Flash 就绪

该位反应了对 Nand Flash 的 RDY 引脚的检测结果

BUSY　　b6　　控制器忙碌标志位：

0：控制器空闲，此时 ALE/CLE 寄存器可以被写入

1：控制器忙，此时 ALE/CLE 寄存器不可被写入，否则，CMDLOSS 标志将被置位，写入的数据将丢失

CMDLOSS　　b5　　命令丢失中断标志位：

读 0：没有发生命令丢失中断

读 1：发生命令丢失中断

写 0：无意义

写 1：清除中断标志

RDMISS	b4	读数据错误中断标志位： 读 0：没有发生读数据错误中断 读 1：发生读数据错误中断 写 0：无意义 写 1：清除中断标志
WRLOSS	b3	写数据丢失中断标志位： 读 0：没有发生写数据丢失中断 读 1：发生写数据丢失中断 写 0：无意义 写 1：清除中断标志
RDYRISE	b2	RDY 信号中断标志位： 读 0：RDY 信号没有上升沿输入 读 1：RDY 信号有上升沿输入 写 0：无意义 写 1：清除中断标志
RD_FULL	b1	读缓冲区满中断标志位： 读 0：读缓冲区未满 读 1：读缓冲区满 该位将在读取 P_NAND_RX_DATA 寄存器后自动清除
WREMPTY	b0	写缓冲区空中断标志位： 读 0：写缓冲区未空 读 1：写缓冲区空 该位将在向 P_NAND_TX_DATA 寄存器写入数据后自动清除

所有 Flash 都存在位翻转的现象，Nand 型 Flash 也不例外。在某些情况下，一个位的数据被发现发生了翻转错误。在大多数应用中，用户需要使用错误更正（ECC）算法来尽量避免错误的发生。SPCE3200 的 Nand 型 Flash 控制器内部具有两个 ECC 计算电路，可以由硬件完成错误更正。下面是对 ECC 电路的控制寄存器的描述。

需要注意的是，两个 ECC 计算电路可以配合针对 512 个字节的数据进行错误更正，对于 2KB 大小页容量的 Nand 型 Flash 芯片，需要将 2KB 的页分为四次来进行计算。

NAND 实际行校验码寄存器：P_NAND_ECC_TRUELP(0x8819001C)

在通过控制器向 Nand Flash 写入数据或从 Nand Flash 读出数据的时候，控制器内部的两个 ECC 计算电路可以计算 512 个字节的数据的第 0~255 字节以及第 256~511 字节的行奇偶校验码，并将计算结果保存至该寄存器内。

在执行写入操作的时候编程者应将该寄存器内容写入 Nand Flash 对应页的备用区域内，以便将



来读取该页数据时可以进行纠错校验。

在执行读取操作的时候，利用该寄存器内的实际数据的奇偶校验值，配合 P_NAND_ECC_TRUECP、P_NAND_ECC_CMPLP 以及 P_NAND_ECC_CMPCP 寄存器的值，ECC 电路可以对数据进行错误校验和纠正。

表 4.156 P_NAND_ECC_TRUELP(0x8819001C)

位	b31~b16	b15~b0
读/写	R/W	R/W
默认值	0	0
名称	TRUELP1	TRUELP0

TRUELP1	b31~b16	第 256~511 字节的实际行奇偶校验码
TRUELP0	b15~b0	第 0~255 字节的实际行奇偶校验码

NAND 实际列校验码寄存器：P_NAND_ECC_TRUECP(0x88190020)

表 4.157 P_NAND_ECC_TRUECP(0x88190020)

位	b11~b6	b5~b0
读/写	R/W	R/W
默认值	111111	111111
名称	TRUECP1	TRUECP0

TRUECP1	b11-b6	第 256~511 字节的实际列奇偶校验码
TRUECP0	b5-b0	第 0~255 字节的实际列奇偶校验码

NAND 对比行校验码寄存器：P_NAND_ECC_CMPLP(0x88190024)

在从 Nand Flash 读取数据时，编程者应该将之前写入备用区域的行奇偶校验码读出并保存至该寄存器，控制器内部的 ECC 电路可以利用该寄存器的值，并配合实际数据的行和列奇偶校验值寄存器 P_NAND_ECC_TRUELP 和 P_NAND_ECC_TRUECP，以及 P_NAND_ECC_CMPCP 寄存器来对数据进行错误校验和纠正。

表 4.158 P_NAND_ECC_CMPLP(0x88190024)

位	b31~b16	b15~b0
读/写	R/W	R/W
默认值	0	0
名称	CALLP1	CALLP0

CALLP1　　b31~b16　　第 256~511 字节的行奇偶校验码

CALLP0　　b15~b0　　第 0~255 字节的行奇偶校验码

NAND 对比列校验码寄存器：P_NAND_ECC_CMPCP(0x88190028)

表 4.159 P_NAND_ECC_CMPCP(0x88190028)

位	**b11~b6**	**b5~b0**
读/写	R/W	R/W
默认值	111111	111111
名称	TRUECP1	TRUECP0

CALCP1　　b11~b6　　第 256~511 字节的列奇偶校验码

CALCP0　　b5~b0　　第 0~255 字节的列奇偶校验码

NAND 校验控制寄存器：P_NAND_ECC_CTRL(0x88190030)

表 4.160 P_NAND_ECC_CTRL(0x88190030)

位	**b2**	**b1**	**b0**
读/写	W	W	W
默认值	0	0	0
名称	CLRECC1	CLRECC0	CALECC

CLRECC1　　b2　　ECC 电路的第 1 部分复位控制位：

写 0：无意义

写 1：复位第 256~511 字节对应的 ECC 计算电路

CLRECC0　　b1　　ECC 电路的第 0 部分复位控制位：

写 0：无意义

写 1：复位第 0~255 字节对应的 ECC 计算电路

CALECC　　b0　　ECC 校验的启动控制位：

写 0：无意义

写 1：ECC 电路将根据 P_NAND_ECC_TRUELP、P_NAND_ECC_TRUECP、P_NAND_ECC_CMPLP 和 P_NAND_ECC_CMPCP 寄存器的值对数据进行错误校验，计算完成后该位将置 0

NAND 校验状态寄存器：P_NAND_ECC_STATUS(0x8819002C)



该寄存器保存了对 0～255 字节和 256～511 字节两部分数据的 ECC 校验结果，用户可以读取该寄存器内容获得错误情况，并根据错误情况做错误校正。

表 4.161 P_NAND_ECC_STATUS(0x8819002C)

位	b31~b30	b29~b27	b26~b24	b23~b16	b15~b14	b13~b11	b10~b8	b7~b0
读/写	R	-	R	R	R	-	R	R
默认值	0	-	0	0	0	-	0	0
名称	ERR1	-	ERRBIT1	ERRBYTE1	ERR0	-	ERRBIT0	ERRBYTE0

ERR1　　b31~b30　　第 256~511 字节的校验结果：

00：没有错误

01：有一处错误，可以纠正（错误位置由 ERRBIT1 和 ERRBYTE1 指出）

10：保留

11：有两处或以上的错误，不可纠正

ERRBIT1　　b26~b24　　ECC 电路检测出的错误字节中的错误位：

000：ERRBYTE1 的第 0 位错误

001：ERRBYTE1 的第 1 位错误

……

111：ERRBYTE1 的第 7 位错误

ERRBYTE1　　b23~b16　　ECC 电路检测出的第 256~511 字节中的错误字节

ERR0　　b15~b14　　第 0~255 字节的校验结果：

00：没有错误

01：有一处错误，可以纠正（错误位置由 ERRBIT0 和 ERRBYTE0 指出）

10：保留

11：有两处或以上的错误，不可纠正

ERRBIT0　　b10~b8　　ECC 电路检测出的错误字节中的错误位：

000：ERRBYTE0 的第 0 位错误

001：ERRBYTE0 的第 1 位错误

……

111：ERRBYTE0 的第 7 位错误

ERRBYTE0　　b7~b0　　ECC 电路检测出的第 0~255 字节中的错误字节

4.13.6 基本操作

Nand 型 Flash 控制器的初始化：

在使用 Nand 型 Flash 控制器之前，必须首先进行初始化。Nand 型 Flash 控制器的初始化比较简单，只需要设置三个寄存器即可：

1) 设置 P_NAND_INTERFACE_SEL 寄存器，选择对应引脚做为 Nand 型 Flash 接口使用；
2) 设置 P_NAND_CLK_CONF 寄存器，使能 Nand 型 Flash 控制器模块时钟；
3) 设置 P_NAND_INT_CTRL 寄存器，根据需要使能中断；

参考代码如下：

```
* P_NAND_INTERFACE_SEL = C_NAND_PORT_SEL;        // 选择引脚为 NAND FLASH
* P_NAND_CLK_CONF = C_NAND_CLK_EN
                  | C_NAND_RST_DIS;              // 使能 NAND FLASH 模块时钟
* P_NAND_INT_CTRL = 0x00000000;                  // 禁止 NAND 所有中断
```

对 Nand 型 Flash 的操作大致包括发送命令、发送地址、写/读数据等。下面针对每种操作叙述寄存器的操作流程。

发送命令：

1) 设置 P_NAND_MODE_CTRL 寄存器，确定 Nand 型 Flash 操作时序。主要考虑的设置项包括：
   a) Flash 模块使能位（FLEN）位必须设置为 1；
   b) CEN 引脚输出设置位（FLCEN）位一般设置为 0，选中 Nand 型 Flash 芯片；
   c) 根据需要设置写保护位（FLWPN）、读写命令位（WRCMD）；
   d) 根据实际连接的 Nand 型 Flash 芯片的容量和地址周期数确定 ALECYC；
   e) 根据实际连接的 Nand 型 Flash 芯片的时序要求设置 RDTYPE、WRLOW、WRHIGH、RDLOW、RDHIGH 等位；
   f) 根据需要设置本次需要传输的数据量 TBYTES。这一步设置是与后面对 Flash 的读写操作相关的，控制器将根据此设定值控制数据写传输数量，同时，ECC 电路也将根据此设定值工作。
2) 向 P_NAND_CLE_COMMAND 寄存器中写入命令；
3) 判断 P_NAND_INT_STATUS 寄存器的 BUSY 位，等待控制器完成操作；

注意，在发送命令的时候由于将同时设置地址周期数量，所以请首先根据实际连接的 Nand 型 Flash 芯片确定其需要的地址周期数。

参考代码如下：

```
* P_NAND_MODE_CTRL = 0x00000001          // NAND 模块使能
                   | 0                   //  片选，CEN 引脚输出低电平
                   | 0x00000004          // WPN 为 1，不保护
                   | 0                   // WRCMD 为 0，读命令
                   | 0x000000C0          // WRLOW 和 RDLOW 设置为 2 个周期
                   | 0x00001100          // WRHIGH 和 RDHIGH 设置为 1 个周期
                   | (4 - 1) << 16;      //  本次传输 4 个字节
```

```
* P_NAND_CLE_COMMAND = 0x90;            // 写入命令 90
while(*P_NAND_INT_STATUS & 0x80);       // 等待控制器完成操作
```

发送地址：

通常根据 Nand 型 Flash 芯片的容量不同，寻址需要的地址周期数也不同。对于不超过四个地址周期的芯片，地址长度不超过 32 位，可以一次写入 P_NAND_ALE_ADDR 寄存器并发送：

1) 设置 P_NAND_MODE_CTRL 寄存器的 ALECYC，选择合适的周期模式，注意不要改变该寄存器其他位的值；
2) 向 P_NAND_ALE_ADDR 寄存器写入 32 位地址（包括 8 位列地址和 10~13 位行地址）；
3) 判断 P_NAND_INT_STATUS 寄存器的 BUSY 位，等待控制器完成操作；

由于在发送命令操作的时候已经设置过 ALECYC 地址周期，如果地址模式没有发生变化，第一步可以省略掉。

参考代码如下：

```
*P_NAND_MODE_CTRL &= 0xFFFFFF3F;          // 将 ALECYC 设置为 0
*P_NAND_MODE_CTRL |= 0x000000C0;          // 选择 4 地址周期
// 上面两步如果与发送命令时的设置一致则可以省略
*P_NAND_ALE_ADDR = address;               // 写入地址
while(*P_NAND_INT_STATUS & 0x80);         // 等待控制器完成操作
```

对于 5 地址周期的寻址模式，寻址地址长度超过 32 位，不能通过一次操作完成。一般可以通过下面的流程实现：

1) 设置 P_NAND_MODE_CTRL 寄存器的 ALECYC，选择两地址周期，注意不要改变寄存器其他位的值；
2) 向 P_NAND_ALE_ADDR 寄存器写入列地址；
3) 判断 P_NAND_INT_STATUS 寄存器的 BUSY 位，等待控制器完成操作；
4) 设置 P_NAND_MODE_CTRL 寄存器的 ALECYC，选择三地址周期，注意不要改变寄存器其他位的值；
5) 向 P_NAND_ALE_ADDR 寄存器写入行地址；
6) 判断 P_NAND_INT_STATUS 寄存器的 BUSY 位，等待控制器完成操作；

参考代码如下：

```
*P_NAND_MODE_CTRL &= 0xFFFFFF3F;          // 将 ALECYC 设置为 0
*P_NAND_MODE_CTRL |= 0x00000040;          // 选择 2 地址周期
*P_NAND_ALE_ADDR = column_address;        // 写入列地址
while(*P_NAND_INT_STATUS & 0x80);         // 等待控制器完成操作
*P_NAND_MODE_CTRL &= 0xFFFFFF3F;          // 将 ALECYC 设置为 0
*P_NAND_MODE_CTRL |= 0x00000080;          // 选择 3 地址周期
```

```
*P_NAND_ALE_ADDR = row_address;            // 写入行地址
while(*P_NAND_INT_STATUS & 0x80);          // 等待控制器完成操作
```

写数据：

向 Flash 写入数据的过程比较简单：

1) 查询 P_NAND_INT_STATUS 寄存器的 WREMPTY 位，等待写缓冲区空；
2) 向 P_NAND_TX_DATA 寄存器写入 32 位数据；
3) 重复这个过程，直至发送完毕 P_NAND_MODE_CTRL 寄存器的 TBYTES 位设置的数据个数；

参考代码如下：512 字节数据，512/4 字数据。

```
for(i=0;i<512/4;i++)
{
    while((*P_NAND_INT_STATUS & 0x01) == 0);       // 等待写缓冲区空
    *P_NAND_TX_DATA = data[i];                     // 写入数据
}
```

读数据：

控制器可以根据 P_NAND_MODE_CTRL 寄存器中的 RDTYPE 以及 TBYTES 的设置，自动获取由 Nand 型 Flash 发送过来的数据。从 Flash 读取数据的过程如下：

1) 查询 P_NAND_INT_STATUS 寄存器的 RDFULL 位，等待读缓冲区满；
2) 从 P_NAND_RX_DATA 寄存器读取接收到的数据；
3) 重复这个过程，直至接收完毕 P_NAND_MODE_CTRL 寄存器的 TBYTES 位设置的数据个数；

参考代码如下：

```
for(i=0; i<512/4; i++)
{
    while((*P_NAND_INT_STATUS & 0x02) == 0);       // 等待读缓冲区空
    data[i] = *P_NAND_RX_DATA;                     // 读取数据
}
```

ECC 校验：

SPCE3200 内置的 ECC 校验码计算电路可以对 512 字节数据进行 ECC 校验码的计算以及纠错比对，如果用户使用的 Nand 型 Flash 芯片的页容量并非 512 字节，则需要以 512 字节为一组分别做 ECC 校验。

使用 ECC 校验，首先需要用户在向 Flash 写入数据的时候计算这些数据的 ECC 校验码，并将这些数据一并保存至 Flash 内，之后需要重新读回数据时，同时也将先前写入的校验码读出，这样，ECC 电路便可以计算实际读取的数据的 ECC 校验码，并与先前写入的校验码比较，以确认读取的



数据是否存在错误，并做错误纠正。

写入数据时 ECC 校验码的生成是 ECC 电路自动完成的，用户在使用时需要遵循如下流程：

1) 启动写操作之前，首先向 P_NAND_ECC_CTRL 寄存器的 CLRECC1 和 CLRECC0 位写入 1，复位内部的两个 ECC 电路；
2) 写操作完成后，读取 P_NAND_ECC_TRUELP 和 P_NAND_ECC_TRUECP 寄存器，得到 ECC 校验值；
3) 根据需要保存 ECC 校验值到 Flash。

参考代码如下：

```
*P_NAND_ECC_CTRL = 0x00000006;              // 写操作前复位 ECC 电路
......
......                                      // 省略 FLASH 写操作过程
uiEcc_Truelp = *P_NAND_ECC_TRUELP;          // 读真实行校验码
uiEcc_Truecp = *P_NAND_ECC_TRUECP;          // 读真实列校验码
......
......                          // 省略 ECC 校验码保存过程，一般保存在 FLASH 备用区域
```

读取数据时 ECC 校验工作需要用户手动控制 ECC 电路进行，应遵循如下流程：

1) 启动读操作之前，首先向 P_NAND_ECC_CTRL 寄存器的 CLRECC1 和 CLRECC0 位写入 1，复位内部的两个 ECC 电路；
2) 读操作完成之后，同时将之前保存在 Flash 内的 ECC 校验值读出；
3) 将读出的 ECC 校验值写入 P_NAND_ECC_CMPLP 和 P_NAND_ECC_CMPCP 寄存器；
4) 向 P_NAND_ECC_CTRL 寄存器的 CALECC 位写入 1，启动 ECC 电路进行校验；
5) 读取 P_NAND_ECC_STATUS 寄存器，判断 512 字节中是否存在错误，并进行纠错工作；

参考代码如下：

```
*P_NAND_ECC_CTRL = 0x00000006;          // 读操作前复位 ECC 电路
......                                  // 省略 FLASH 读操作过程
......
// 需要将之前保存的 ECC 校验码重新读取，一般从 flash 备用区域读取，省略
*P_NAND_ECC_CMPLP = uiEcc_Cmplp;        // 写入对比行校验码
*P_NAND_ECC_CMPCP = uiEcc_Cmpcp;        // 写入对比列校验码
*P_NAND_ECC_CTRL = 1;                   // 启动 ECC 电路进行校验
uiEcc_Result = *P_NAND_ECC_STATUS;      // 读取校验结果
// 这里省略根据校验结果进行纠错的过程
......
```

4.14 SD 卡控制器

4.14.1 概述

安全数字（SD，Secure Digital）存储器卡实际是一种闪存（Flash）卡，专门为满足安全使用、大容量、高性能以及作为时尚音视频电子产品内部需要的存储配件而设计的。SD 存储卡的通讯通过一种新式 9 针串行接口实现，可工作在一个低电压范围内。SD 卡使用的 9 针接口除去电源和地，包括以下信号线：

- CLK：从主机到 SD 卡的时钟信号，用于实现同步串行通信和数据传输
- CMD：双向的命令/响应信号线。主机通过此线向 SD 卡串行发送命令或通过此线串行接收 SD 卡的响应
- DAT0 ~ DAT3：4 位双向数据总线。主机通过该总线向 SD 卡写入数据或从 SD 卡读取数据

SD 卡内部的存储区域是以扇区为单位管理的。在对存储区域的数据进行擦除、写和读操作时，必须以扇区为单位进行。通常 SD 卡的每个扇区为 512 字节。

主机并不能直接读写 SD 卡内部的数据，而是依靠命令控制 SD 卡进行操作。所以，主机与 SD 卡之间的通信分为命令比特流和数据比特流两种形式，使用起始位和结束位来标识一个完整的比特流。

- 命令：由主机通过 CMD 线发送给 SD 卡，用以告知 SD 卡启动什么样的操作
- 响应：在接到主机的命令后由 SD 卡回送给主机的应答信息。SD 卡在接到主机的命令后大多数情况下会返回应答信息给主机。响应数据也是通过 CMD 线传输的
- 数据：向 SD 卡写入或从 SD 卡读出的用户数据，通过 4 位宽度的数据总线传输在主机和 SD 卡之间传输

主机与 SD 卡之间最基本的通信是命令/响应传输（如图 4.83 所示）。这种通信方式可以直接在主机与 SD 卡之间传递信息。

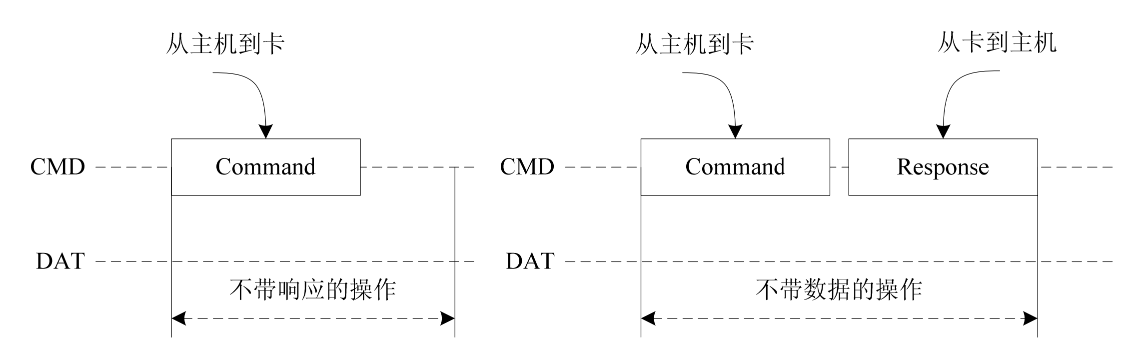


图 4.83 命令/响应传输操作

有些命令将会带有数据的传输，比如读/写扇区命令等。这些数据在 SD 卡接到命令之后开始通过 4 位宽度的数据总线传输。图 4.84 所示的是主机读取 SD 卡内部的扇区数据的操作，图 4.85 所示的是主机向 SD 卡写入扇区数据的操作。可以看到，在基本的命令/响应传输的基础上，配合 4 位宽度的数据总线附加进行数据流的传输。

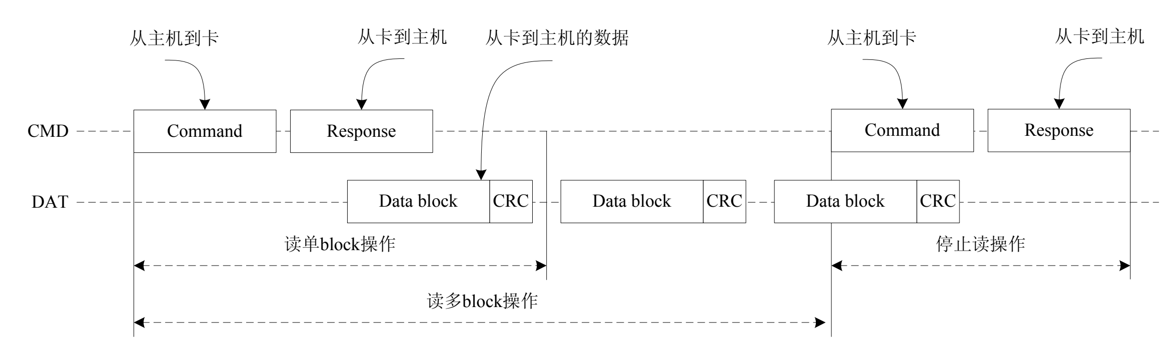


图 4.84 读扇区操作

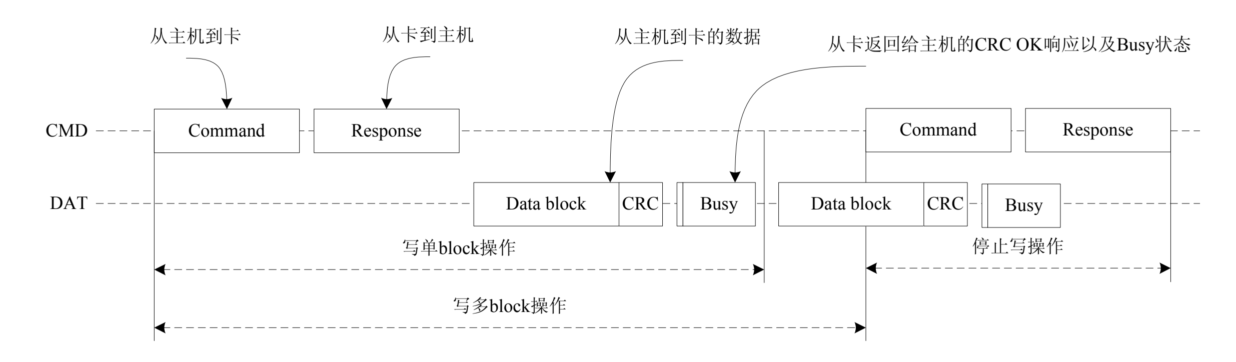


图 4.85 写扇区操作

SPCE3200 内部的 SD 卡控制器可以通过编程控制完成以上所有四种基本通信功能，并能够提供通过 DMA 访问的高速传输性能，从而获得产品的最佳性价比。

4.14.2 特性

SPCE3200 的 SD 卡控制器具有如下特点：

- 完全兼容 SD 存储卡的规格
- 能够直接接收 SD 命令，提高了兼容性
- 可编程选择 SD 总线的时钟速度
- 提供在缓存区满状态下 SD 总线的时钟控制
- 提供中断配置
- 支持 DMA 读/写操作
- 支持 1 位 / 4 位 SD 工作模式
- 支持对 SD I/O 卡的中断检测。SD I/O 卡是设计在 SD 存储卡的基础上并能与其兼容的，其设计的目的是为了给那些移动电子设备提供高速且低功耗的数据输入/输出功能。

4.14.3 引脚描述

SPCE3200 的 SD 卡接口与 SPI 和 NAND 型 FLASH 接口复用，使用时需要设置相关寄存器以便使其工作在 SD 接口模式。SD 卡接口的引脚描述如表 4.114 所示。

表 4.162 SD 卡接口引脚对应表

引脚名称	引脚号	引脚属性	引脚功能
SD_CLK	126	O	SD 接口的 CLK 输出引脚

SD_CMD	125	I/O	SD 接口的 CMD 引脚
SD_D[0]	117	I/O	SD 接口的数据线的 Bit0
SD_D[1]	116	I/O	SD 接口的数据线的 Bit1
SD_D[2]	115	I/O	SD 接口的数据线的 Bit2
SD_D[3]	112	I/O	SD 接口的数据线的 Bit3

4.14.4 结构

SD 卡控制器的结构如图 4.86 所示。

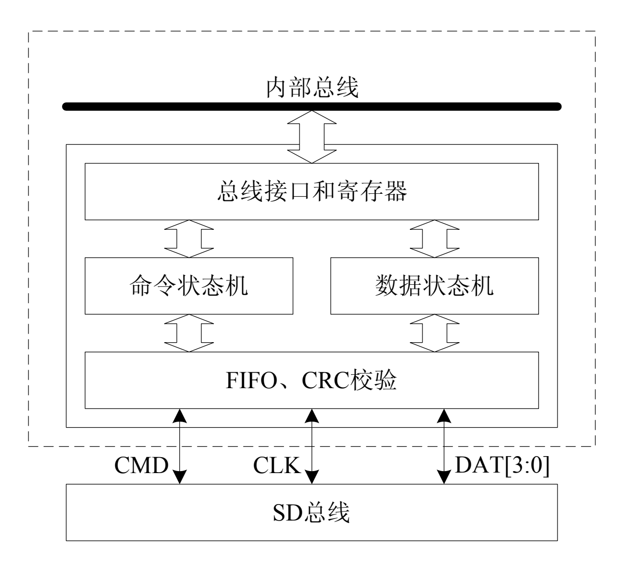


图 4.86 SD 卡控制器结构

4.14.5 寄存器描述

SD 卡控制器共有 10 个寄存器，如表 4.163 所示。通过对这 10 个寄存器的操作，即可使用 SD 卡控制器扩展 SD 卡存储设备。

表 4.163 SD 卡控制器相关寄存器列表

寄存器名称	助记符	地址
SD 接口选择寄存器	P_SD_INTERFACE_SEL	0x882000A4
SD 时钟配置寄存器	P_SD_CLK_CONF	0x882100A4
SD 控制寄存器	P_SD_MODE_CTRL	0x88180018
SD 中断控制寄存器	P_SD_INT_CTRL	0x8818001C
SD 中断状态寄存器	P_SD_INT_STATUS	0x88180014
SD 命令设置寄存器	P_SD_COMMAND_SETUP	0x88180008
SD 参数数据寄存器	P_SD_ARGUMENT_DATA	0x8818000C
SD 响应数据寄存器	P_SD_RESPONSE_DATA	0x88180010

SD 发送数据寄存器	P_SD_TX_DATA	0x88180000
SD 接收数据寄存器	P_SD_RX_DATA	0x88180004

SD 接口选择寄存器：P_SD_INTERFACE_SEL(0x882000A4)

SPCE3200 的 SD 卡接口与 SPI 和 NAND 型 FLASH 接口复用，使用时需要设置 SD 接口选择寄存器以便使其工作在 SD 接口模式。

表 4.164 P_SD_INTERFACE_SEL(0x882000A4)

位	b16	b15-b0
读/写	R/W	-
默认值	0	-
名称	SD_EN	-

SD_EN　　b16　　SD 卡接口使能位：
0：不使能为 SD 卡接口
1：使能为 SD 卡接口

SD 时钟配置寄存器：P_SD_CLK_CONF(0x882100A4)

SD 时钟配置寄存器用于使能或禁止 SD 控制器模块时钟，或软复位 SD 卡控制器模块。

表 4.165 P_SD_CLK_CONF(0x882100A4)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	SD_RST	SD_STOP

SD_RST　　b1　　SD 模块时钟复位位：
0：SD 模块时钟复位
1：SD 模块时钟不复位

SD_STOP　　b0　　SD 模块时钟停止位：
0：SD 模块时钟停止
1：SD 模块时钟使能

SD 控制寄存器：P_SD_MODE_CTRL(0x88180018)

SD 控制寄存器用于控制 SD 总线的时钟频率、扇区大小等工作特性。需要注意的是，该寄存器仅当 SD 卡控制器的 Status 寄存器的 BUSY 位为 0 时才可以被改变。

表 4.166 P_SD_MODE_CTRL(0x88180018)

位	b31~b28			b27~b16		
读/写	-			W		
默认值	-			0		
名称	-			BLKLEN		
位	b15~b12	b11	b10	b9	b8	b7~b0
读/写	-	W	W	W	W	W
默认值	-	0	0	0	0	0x54
名称	-	EN_SD	IOEN	DMAMODE	BUSWIDTH	CLKDIV

BLKLEN　b27~b16　SD 卡的扇区长度。以字节为单位。该值应该与实际连接的 SD 卡的扇区长度相等

EN_SD　b11　SD 总线使能控制。该位控制 IO 口是否做为 SD 总线使用还是做为 GPIO 使用。如果希望使用 SD 卡控制器，该位必须设置为 1。

0：SD_CLK、SD_CMD、SD_DAT0 做为 GPIO

1：SD_CLK、SD_CMD、SD_DAT0 做为 SD 总线

另外，如果 BUSWIDTH 位设置为 1，当 EN_SD 位为 1 时 SD_DAT1、SD_DAT2、SD_DAT3 也做为 SD 总线使用

IOEN　b10　I/O 卡中断使能：

0：禁止 I/O 卡中断

1：使能 I/O 卡中断

DMAMODE　b9　DMA 传输方式选择位：

0：不使用 DMA 传输方式

1：使用 DMA 传输方式

BUSWIDTH　b8　SD 数据总线宽度设置位：

0：1 位宽度

1：4 位宽度

CLKDIV　b7~b0　SD 总线的时钟频率设置，SD 总线的时钟频率计算如下：$f_{SDCLK}=f_{SYSCLK} / 2(CLKDIV+1)$。CLKDIV 的默认值为 0x54

SD 中断控制寄存器：P_SD_INT_CTRL(0x8818001C)

SD 中断控制寄存器用于使能或禁止 SD 卡控制器的中断。SD 卡控制器使用 IRQ40 中断。

表 4.167 P_SD_INT_CTRL(0x8818001C)

位	b31~b7	b6	b5	b4	b3	b2	b1	b0
读/写	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W
默认值	-	0	0	0	0	0	0	0
名称	-	INTEN6	INTEN5	INTEN4	INTEN3	INTEN2	INTEN1	INTEN0

INTEN6　　b6　　I/O 卡中断使能位：

0：不使能 I/O 卡中断

1：使能 I/O 卡中断

INTEN5　　b5　　卡插入/拔出中断使能位：

0：不使能卡插入/拔出中断

1：使能卡插入/拔出中断

INTEN4　　b4　　数据缓冲区清空中断使能位：

0：不使能数据缓冲区清空中断

1：使能数据缓冲区清空中断

INTEN3　　b3　　数据缓冲区溢出中断使能位：

0：不使能数据缓冲区溢出中断

1：使能数据缓冲区溢出中断

INTEN2　　b2　　命令缓冲区溢出中断使能位：

0：不使能命令缓冲区溢出中断

1：使能命令缓冲区溢出中断

INTEN1　　b1　　数据传输完成中断使能位：

0：不使能数据传输完成中断

1：使能数据传输完成中断

INTEN0　　b0　　命令发送完成中断使能位：

0：不使能命令发送完成中断

1：使能命令发送完成中断

SD 中断状态寄存器：P_SD_INT_STATUS(0x88180014)

读取 SD 中断状态寄存器的内容可以判断 SD 卡控制器的状态。

表 4.168 P_SD_INT_STATUS(0x88180014)

位	b31~b14	b13	b12	b11	b10	b9	b8
读/写	-	R	R	-	R	R	R
默认值	-	0	0	-	0	0	0
名称	-	CARDINT	CARDPRE	-	DATCRCERR	TIMEOUT	DATBUFEMPTY

位	b7	b6	b5	b4	b3	b2	b1	b0
读/写	R	R	R	R	R	R	R	R
默认值	0	0	0	0	0	0	0	0
名称	DATBUF FULL	CMDBUF FULL	RSPCRCERR	RSPIDXERR	DATCOM	CMDCOM	CARDBUSY	BUSY

CARDINT	b13	I/O 卡中断挂起标志位： 0：没有发生 I/O 卡挂起中断 1：发生 I/O 卡挂起中断 在 P_SD_MODE_CTRL 寄存器中的 IOEN 位为 1 时这一位才会被置位。主机需要通过设备特殊命令来清除该中断标志，将这一位写为 1 没有意义
CARDPRE	b12	SD 卡是否插入： 0：SD 卡未插入 1：SD 卡插入 这一位仅在 SD 卡控制器空闲时检测 SD 接口的 DAT3 管脚，控制器的工作不会受此位的影响，且主机不论这一位为何值都可以启动数据传输。 将此位写 1 会清除卡插入挂起的中断状态标志。
DATCRCERR	b10	接收数据的CRC校验码错误，或发送数据后SD卡返回CRC错误响应 0：没有发生CRC错误 1：发生CRC错误
TIMEOUT	b9	命令响应超时、读数据响应超时或发送完写命令后一段时间内没有数据写入 P_SD_TX_DATA 而引起的超时。 0：没有超时 1：发生超时
DATBUFEMPTY	b8	当存储在 FIFO 中的数据数目未超过特定数目（trigger level）时这一位会置 1，表示数据缓冲区空。 向 P_SD_TX_DATA 寄存器写入合适数目的数据，或将 P_SD_COMMAND_SETUP 寄存器中的 STPCMD 位置 1 时会将这一位清 0。
DATBUFFULL	b7	当存储在 FIFO 中的数据数目超过特定数目（trigger level）时这一位会置 1，表示数据缓冲区满。 清除该标志的方法：从 P_SD_RX_DATA 寄存器读出合适数目的数据；将 P_SD_COMMAND_SETUP 寄存器中的 STPCMD 位置 1 时会将这一位清 0。



CMDBUFFULL　　b6　　命令响应寄存器 P_SD_RESPONSE_DATA 满则该位被置 1。

清除该标志的方法：读取 P_SD_RESPONSE_DATA 的值

启动一个新的传输；将 P_SD_COMMAND_SETUP 寄存器中的 STPCMD 位置 1。

RSPCRCERR　　b5　　响应数据的 CRC 校验码错误则该位被置 1。

在 R3 型响应情况下接收到的 CRC 校验码不为 6b'111111 则该位被置 1

RSPIDXERR　　b4　　响应中的索引是否出错：

0：正常

1：出错

DATCOM　　b3　　数据传输完毕标志位：

0：数据传输未完成

1：数据传输完毕

CMDCOM　　b2　　命令传输完毕标志位：

0：命令传输未完成

1：命令传输完毕

CARDBUSY　　b1　　SD 卡是否 BUSY（DAT0 是否为低电平）：

0：SD 卡不忙

1：SD 卡忙

注意：主机在发送写命令后需要查询此位

BUSY　　b0　　SD 卡控制器忙碌标志位：

0：控制器空闲

1：控制器忙碌

SD 命令设置寄存器：P_SD_COMMAND_SETUP(0x88180008)

主机通过 SD 命令设置寄存器控制 SD 卡控制器的操作，SD 卡控制器通过读取这个寄存器的值决定是否向 SD 卡发送命令。

表 4.169 P_SD_COMMAND_SETUP(0x88180008)

位	b14~b12	b11	b10	b9	b8	b7	b6	b5~b0
读/写	W	W	W	W	W	W	W	W
默认值	0	0	0	0	0	0	0	0
名称	RESPTYPE	INICARD	MULBLK	TxData	CMDWD	RUNCMD	STPCMD	CMDCODE

RESPTYPE　　b14~b12　　命令的响应类型：

		000：没有响应　　001：R1 类型的响应 010：R2 类型的响应　011：R3 类型的响应 110：R6 类型的响应　111：R1b 类型的响应 目前只有 R2 类型具有 128bit 的响应数据长度，其他的响应类型均为 32bit 响应数据长度
INICARD	b11	向该位写 1 将使 SD 卡控制器通过 SD 总线的时钟线发送 74 个脉冲
MULBLK	b10	当前命令是否为多扇区操作命令： 0：单扇区操作命令 1：多扇区操作命令
TxData	b9	当前命令是否需要发送/接收数据： 0：需要接收数据（Read 操作） 1：需要发送数据（Write 操作） 注意：该位仅当下面的 CMDWD 位设置为 1 时有效
CMDWD	b8	当前命令是否伴随有数据传输： 0：没有数据传输 1：有数据传输
RUNCMD	b7	将该位写为 1 将启动 SD 卡控制器按照当前的配置发送命令给 SD 卡。 该位将在 SD 卡控制器启动命令传输后自动清 0，但是，用户必须等待 P_SD_INT_STATUS 寄存器的 BUSY 位为 0 时启动下一次命令传输。
STPCMD	b6	向该位写 1 将迫使 SD 卡控制器回到 IDLE 状态。当 SD 卡控制器回到 IDLE 状态后，该位自动清 0。
CMDCODE	b5~b0	主机要发送的命令序号

SD 参数数据寄存器：P_SD_ARGUMENT_DATA(0x8818000C)

主机把需要传递给 SD 卡的参数写入 SD 参数数据寄存器。SD 卡控制器将此寄存器内容做为命令参数发送给 SD 卡。

表 4.170 P_SD_ARGUMENT_DATA(0x8818000C)

位	b31-b0
读/写	W
默认值	0
名称	Argument



Argument　　b31~b0　　传输给 SD 卡的命令参数，有效位为 b15-b0

SD 响应数据寄存器：P_SD_RESPONSE_DATA(0x88180010)

由 SD 卡返回的响应将被保存在该寄存器内。注意，SD 卡控制器并不关心该寄存器内的响应数据的内容和意义，但是主机必须对此关注。

R1、R1b、R3、R6 型响应具有 6 位的命令索引和 32 位的响应数据，而 R2 型响应具有 128 位的响应数据。主机需要查询 P_SD_INT_STATUS 寄存器的 CMDBUFFULL 位，当该位为 1 时才可以读取该寄存器内的响应数据。

表 4.171 P_SD_RESPONSE_DATA(0x88180010)

位	**b31~b0**
读/写	R
默认值	0
名称	Response

Response　　b31~b0　　来自 SD 卡的响应数据

SD 发送数据寄存器：P_SD_TX_DATA(0x88180000)

当主机写入 32 位数据到此寄存器后，SD 卡控制器就会将该数据发送给 SD 卡。而当数据发送完毕后，状态寄存器 P_SD_INT_STATUS 中的 DATBUFEMPTY 位会置 1，或产生 DMA 传输请求。而向此寄存器写入数据，也必须等待 P_SD_INT_STATUS 寄存器中的 DATBUFEMPTY 位为 1。

表 4.172 P_SD_TX_DATA(0x88180000)

位	**b31-b0**
读/写	W
默认值	0
名称	DataTx

DataTx　　b31~b0　　需要发送给 SD 卡的数据，需等待 P_SD_INT_STATUS 寄存器的 DATBUFEMPTY 位为 1 时向该寄存器写入数据

SD 接收数据寄存器：P_SD_RX_DATA(0x88180004)

从 SD 卡读出的数据要存放在这个寄存器里。当由 SD 卡接收到 32 位数据时，状态寄存器 P_SD_INT_STATUS 中的 DATBUFFULL 位会置 1，或会产生 DMA 请求。注意，只有在 DATBUFFULL 位为 1 时才能从此寄存器里读出数据。

表 4.173 P_SD_RX_DATA(0x88180004)

位	**b31-b0**

读/写	R
默认值	0
名称	DataRx

DataRx　　b31-b0　　从 SD 卡接收到的数据，需等待 P_SD_INT_STATUS 寄存器的 DATBUFFULL 位为 1 时才能从该寄存器读取数据

4.14.6 基本操作

SD 卡控制器的初始化：

在使用 SD 卡控制器之前，必须对其进行初始化，设置其工作模式。

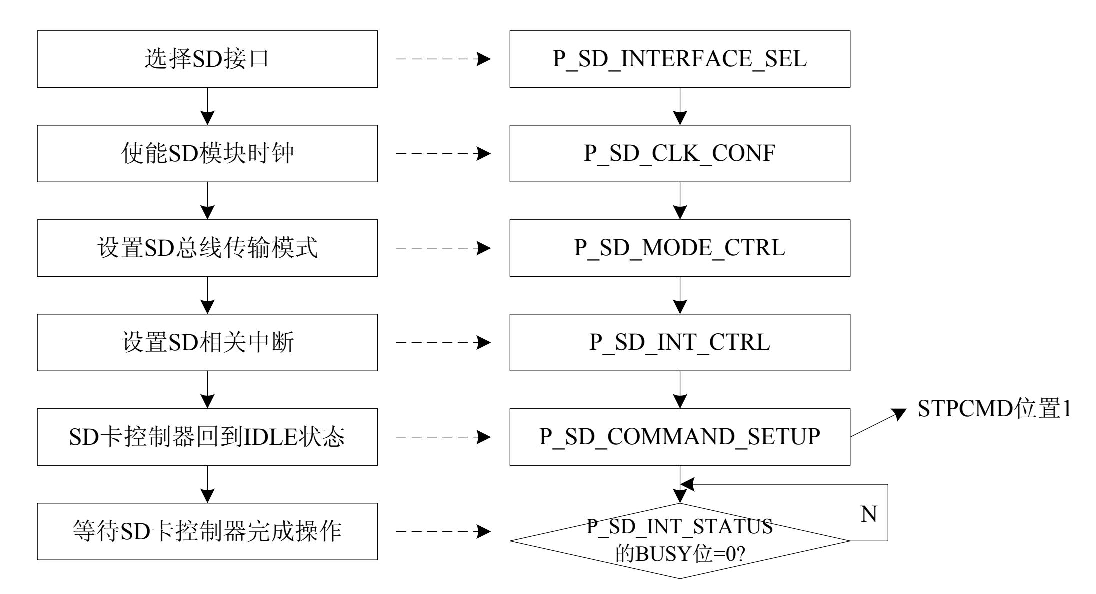


图 4.87 SD 卡控制器初始化步骤

1) 设置 P_SD_INTERFACE_SEL 寄存器，选择 SD 模块控制器接口；
2) 设置 P_SD_CLK_CONF 寄存器，使能 SD 卡控制器模块时钟；
3) 设置 P_SD_MODE_CTRL 寄存器，设定 SD 卡的扇区大小、使能 SD 卡控制器、根据需要打开或者关闭 DMA 传输方式、根据需要设置 SD 数据总线宽度；
4) 设置 P_SD_INT_CTRL 寄存器，根据需要打开或关闭 SD 卡相关中断；
5) 设置 P_SD_COMMAND_SETUP 寄存器的 STPCMD 位为 1，使 SD 卡控制器回到 IDLE 状态；
6) 读取 P_SD_INT_STATUS 寄存器，等待 BUSY 位为 0。

参考代码如下：

```
*P_SD_INTERFACE_SEL = C_SD_PORT_SEL;              // 选择复用端口为 SD 接口
*P_SD_CLK_CONF = C_SD_CLK_EN | C_SD_RST_DIS;      // 使能 SD 模块时钟
*P_SD_MODE_CTRL = (512 << 16)                     // 设置 SD 卡扇区为 512 字节
```



```
                     | C_SD_BUS_4BIT                       // 设置 SD 数据总线为 4bit
                     | C_SD_PORT_EN;                       // 使能 SD 控制器
*P_SD_INT_CTRL = C_SD_CARDDETECT_INTEN;                    // 使能 SD 卡卡插入检测中断
*P_SD_COMMAND_SETUP = C_SD_74CLK_START;                    // 初始化
while(*P_SD_INT_STATUS & C_SD_CTRL_BUSY);                  // 等待 SD 控制器完成操作
```

接下来，便可以使用 SD 卡控制器向 SD 卡发送命令，控制 SD 卡的读写操作。下面以发送命令为例，说明控制 SD 卡控制器发送命令和数据传输的一般步骤。

主机向 SD 卡发送的命令大致有如下四种类型：不带响应的命令（基本操作，用于控制 SD 卡的动作）、不带数据的命令（基本操作，用于控制 SD 卡的动作并从 SD 卡读取基本信息）、带数据的读操作命令（用于从 SD 卡读取数据）、带数据的写操作命令（用于向 SD 卡写入数据）。分别说明使用 SD 卡控制器实现四种操作的过程：

不带响应的命令

SD 卡的命令集中有些命令，SD 卡是不需要返回响应给主机的，如命令序号为 0 的迫使 SD 卡回到 IDLE 状态命令。使用这类命令对应的寄存器操作顺序为：

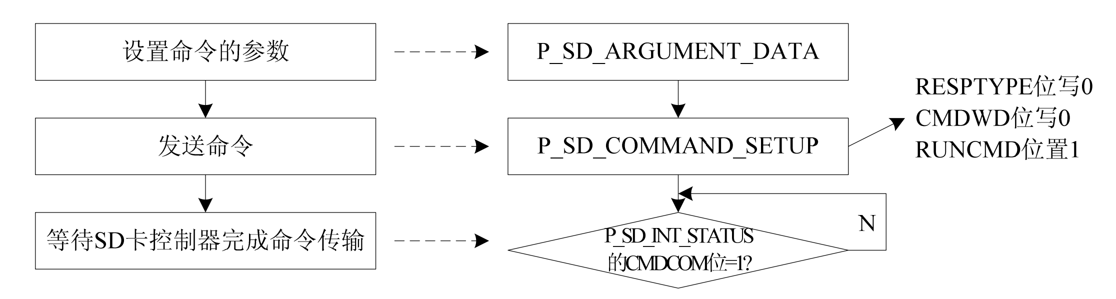


图 4.88 使用不带响应的命令的操作步骤

1) 首先将命令参数写入 P_SD_ARGUMENT_DATA 寄存器；
2) 向 P_SD_COMMAND_SETUP 寄存器写入命令序号，同时，
   a) 将 RESPTYPE 对应位设置为 0，表示该命令不带有响应数据；
   b) 将 CMDWD 位设置为 0，表示该命令不带有数据传输；
   c) 将 RUNCMD 位设置为 1，以便在写入命令序号后立即启动 SD 卡控制器的命令传输；
3) 查询 P_SD_INT_STATUS 寄存器的 CMDCOM 位，等待该位为 1，此时 SD 卡控制器的命令传输操作完成。

参考代码如下：

```
// 以发送 CMD0 命令为例
*P_SD_ARGUMENT_DATA = 0;                                // 该命令没有参数
*P_SD_COMMAND_SETUP = C_SD_RSP_R0                       // 没有响应
                    | C_SD_CMD_WITHOUTDATA              // 没有参数
                    | C_SD_CMD_START                    // 立即发送
                    | 0;                                // SD 命令 0，CMD0
while((*P_SD_INT_STATUS & C_SD_CMD_COMPLETE) == 0);     // 等待命令传输完成
```

不带数据的命令

SD 卡的命令集中有一些命令需要 SD 卡返回响应给主机，但是不需要有其他数据传输的过程。比如命令序号为 9 的读取 SD 卡 CSD 寄存器的命令。使用这类命令对应的寄存器操作顺序为：

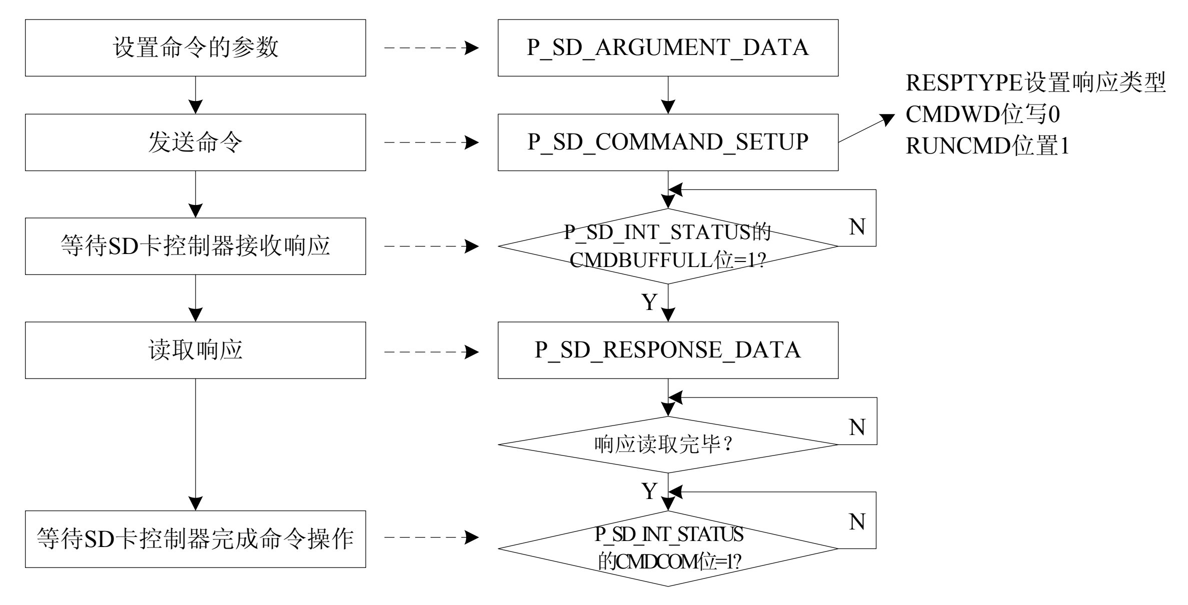


图 4.89 使用不带数据的命令的操作步骤

1) 首先将命令参数写入 P_SD_ARGUMENT_DATA 寄存器；
2) 向 P_SD_COMMAND_SETUP 寄存器写入命令序号，同时，
   a) 将 RESPTYPE 对应位设置为该命令对应的响应类型；
   b) 将 CMDWD 位设置为 0，表示该命令不带有数据传输；
   c) 将 RUNCMD 位设置为 1，以便在写入命令序号后立即启动 SD 卡控制器的命令传输；
3) 查询 P_SD_INT_STATUS 寄存器的 CMDBUFFULL 位，等待该位为 1，此时 SD 卡控制器接收到 32 位长度的响应数据；
4) 读取 P_SD_RESPONSE_DATA 寄存器，得到 SD 卡发送回来的响应；
5) 如果响应数据不止 32 位，则跳至第 3 步，重复读取过程，直至所有响应读取完毕；
6) 查询 P_SD_INT_STATUS 寄存器的 CMDCOM 位，等待该位为 1，此时 SD 卡控制器的命令传输操作完成。

参考代码如下：

```
// 以发送 CMD9 查询 SD 卡的 CSD 寄存器命令为例
*P_SD_COMMAND_SETUP = arg;                          // 写入命令参数
*P_SD_COMMAND_SETUP = C_SD_CMD_WITHOUTDATA          // 没有数据
                      | C_SD_RSP_R2                 // 响应类型为 R2
                      | C_SD_CMD_START              // 立即发送
                      | 9;                          // SD 命令 9，CMD9
for(i=0; i<4; i++)                                  // 接收 128 位响应数据
{
```

```
        while((*P_SD_INT_STATUS & C_SD_CMDBUF_FULL) == 0);     // 等待响应寄存器满
        response[i] = *P_SD_RESPONSE_DATA;                     // 读取响应
    }
    while((*P_SD_INT_STATUS & C_SD_CMD_COMPLETE) == 0);        // 等待命令传输完成
```

带数据的读操作命令

当主机需要读取 SD 卡的扇区时，需要使用带数据的读操作命令。使用这类命令对应的寄存器操作顺序为：

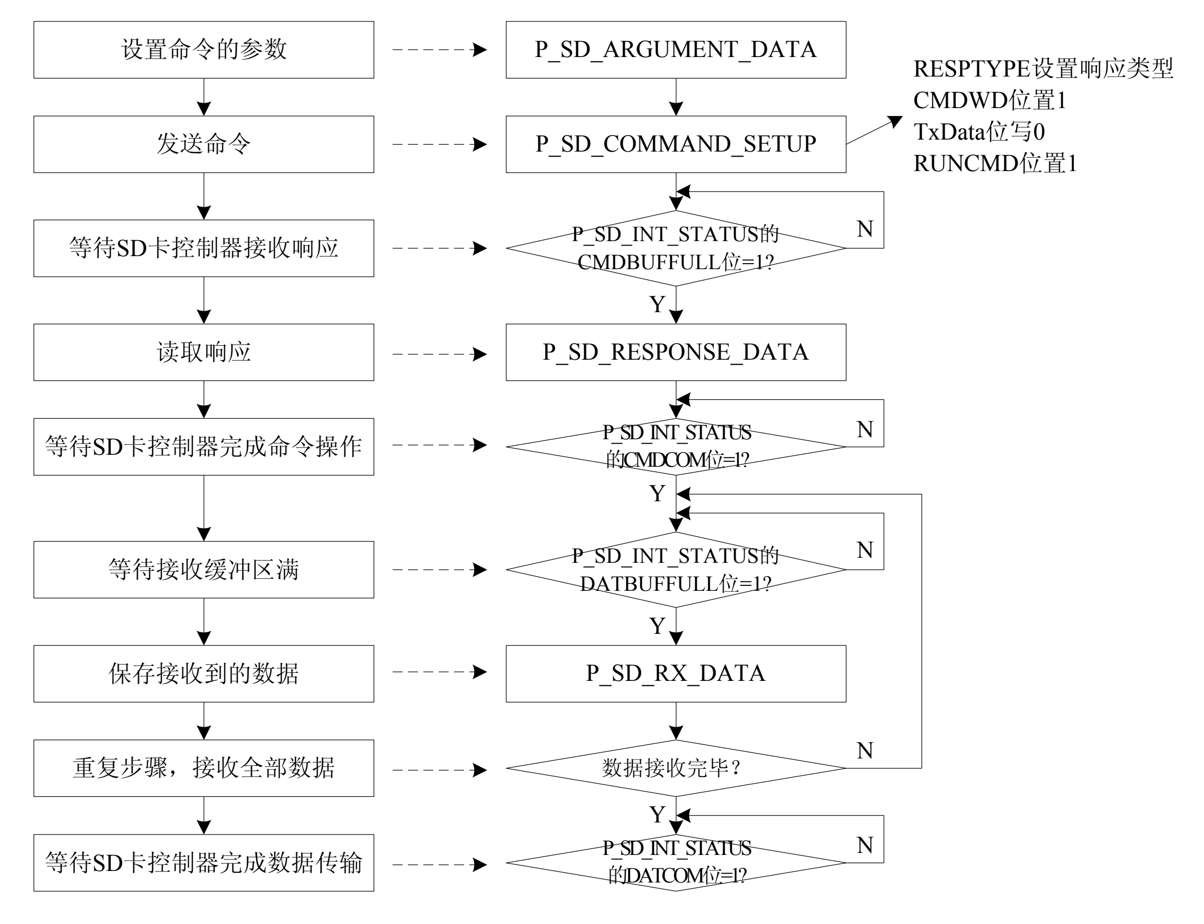


图 4.90 使用带数据的读操作命令的操作步骤

1) 首先将命令参数写入 P_SD_ARGUMENT_DATA 寄存器；
2) 向 P_SD_COMMAND_SETUP 寄存器写入命令序号，同时，
   a) 将 RESPTYPE 对应位设置为该命令对应的响应类型；
   b) 将 CMDWD 位设置为 1，表示该命令带有数据传输；
   c) 将 TxData 位设置为 0，表示该命令需要读取 SD 卡的数据；
   d) 将 RUNCMD 位设置为 1，以便在写入命令序号后立即启动 SD 卡控制器的命令传输；
3) 查询 P_SD_INT_STATUS 寄存器的 CMDBUFFULL 位，等待该位为 1，此时 SD 卡控制器接收到 32 位长度的响应数据；
4) 读取 P_SD_RESPONSE_DATA 寄存器，保存 SD 卡发送回来的响应（一般这类命令只有 32

位的响应）；

5) 查询 P_SD_INT_STATUS 寄存器的 CMDCOM 位，等待该位为 1，此时 SD 卡控制器的命令传输操作完成；
6) 查询 P_SD_INT_STATUS 寄存器的 DATBUFFULL 位，等待该位为 1，此时 SD 卡控制器接收到 4 字节的数据；
7) 从 P_SD_RX_DATA 寄存器读取接收到的数据并保存；
8) 跳至第 6 步，重复接收数据过程，直至接收完毕整个扇区的数据；
9) 查询 P_SD_INT_STATUS 寄存器的 DATCOM 位，等待该位为 1，此时 SD 卡控制器完成数据接收工作。

参考代码如下：

```
// 以发送 CMD17 读扇区命令为例
*P_SD_ARGUMENT_DATA = blockaddress;                      // 写入命令参数，扇区地址
*P_SD_COMMAND_SETUP = C_SD_CMD_WITHDATA                  // 带数据
                    | C_SD_DATA_RECEIVE                  // 数据接收
                    | C_SD_RSP_R1                        // 响应类型为 R1
                    | C_SD_CMD_START                     // 立即发送
                    | 17;                                // SD 命令 17，CMD17
while((*P_SD_INT_STATUS & C_SD_CMDBUF_FULL) == 0);       // 等待响应寄存器满
response = *P_SD_RESPONSE_DATA;                          // 保存响应数据
while((*P_SD_INT_STATUS & C_SD_CMD_COMPLETE) == 0);      // 等待命令传输完毕
for(i=0; i<512/4; i++)                                   // 接收一个扇区 512 字节数据
{
    while((*P_SD_INT_STATUS & C_SD_DATABUF_FULL) == 0);// 等待接收缓冲区满
    BlockData[i] = *P_SD_RX_DATA;                        // 以字为单位保存收到的数据
}
while((*P_SD_INT_STATUS & C_SD_DATA_COMPLETE) == 0); // 等待数据传送完毕
```

带数据的写操作命令

当主机需要向 SD 卡的扇区写入数据时，需要使用带数据的写操作命令。使用这类命令对应的寄存器操作顺序为：

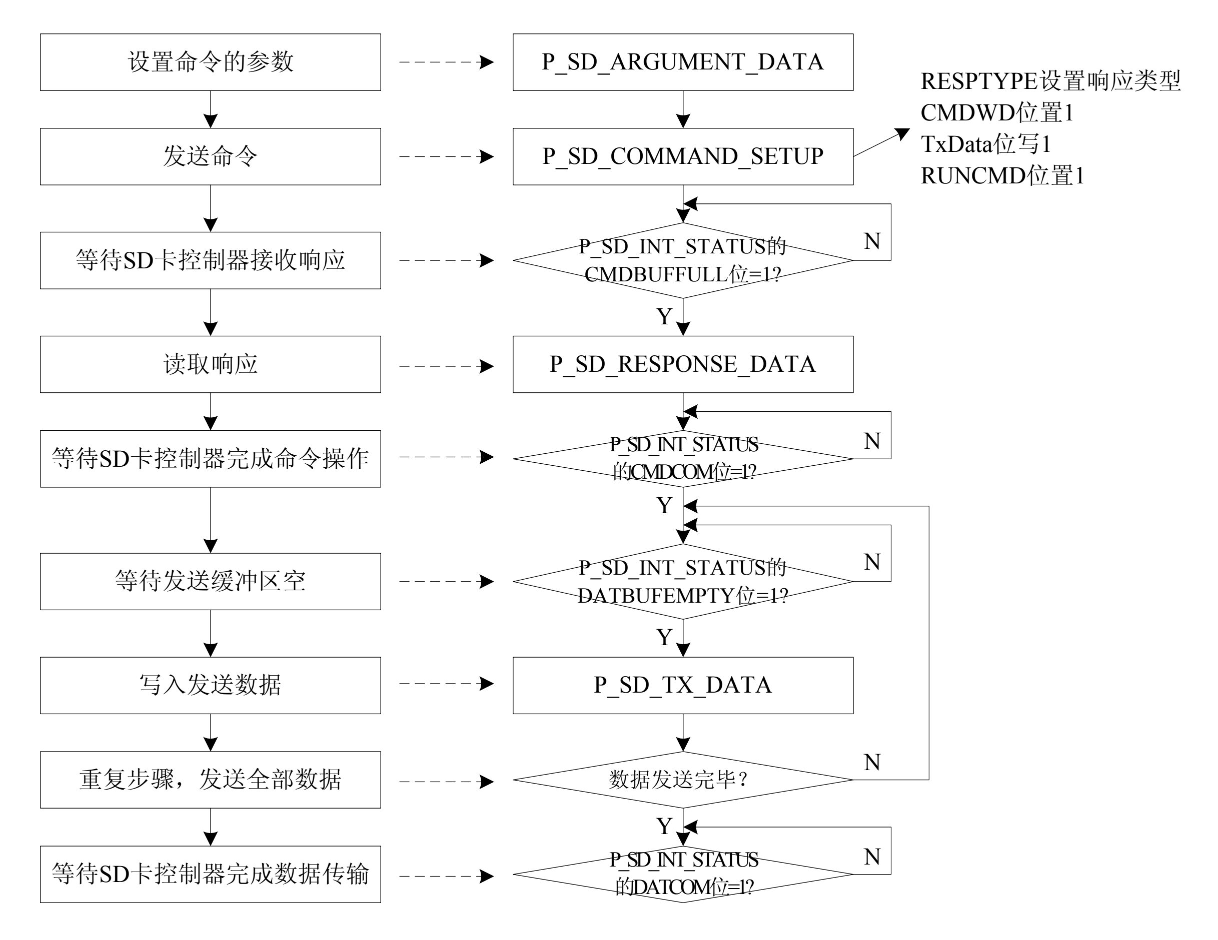


图 4.91 使用带数据的写操作命令的操作步骤

1) 首先将命令参数写入 P_SD_ARGUMENT_DATA 寄存器；
2) 向 P_SD_COMMAND_SETUP 寄存器写入命令序号，同时，
   a) 将 RESPTYPE 对应位设置为该命令对应的响应类型；
   b) 将 CMDWD 位设置为 1，表示该命令带有数据传输；
   c) 将 TxData 位设置为 1，表示该命令需要向 SD 卡写入数据；
   d) 将 RUNCMD 位设置为 1，以便在写入命令序号后立即启动 SD 卡控制器的命令传输；
3) 查询 P_SD_INT_STATUS 寄存器的 CMDBUFFULL 位，等待该位为 1，此时 SD 卡控制器接收到 32 位长度的响应数据；
4) 读取 P_SD_RESPONSE_DATA 寄存器，保存 SD 卡发送回来的响应（一般这类命令只有 32 位的响应）；
5) 查询 P_SD_INT_STATUS 寄存器的 CMDCOM 位，等待该位为 1，此时 SD 卡控制器的命令传输操作完成；
6) 查询 P_SD_INT_STATUS 寄存器的 DATBUFEMPTY 位，等待该位为 1，此时 SD 卡发送缓存区为空；
7) 向 P_SD_TX_DATA 寄存器写入数据；
8) 跳至第 6 步，重复发送数据过程，直至将整个扇区的数据发送完毕；
9) 查询 P_SD_INT_STATUS 寄存器的 DATCOM 位，等待该位为 1，此时 SD 卡控制器完成数据接收工作。

参考代码如下：

```
// 以发送 CMD24 写单扇区命令为例
*P_SD_ARGUMENT_DATA = blockaddress;                         // 写入命令参数，扇区地址
*P_SD_COMMAND_SETUP = C_SD_CMD_WITHDATA                     // 带数据
                    | C_SD_DATA_TRANFER                     // 数据发送
                    | C_SD_RSP_R1                           // 响应类型为 R1
                    | C_SD_CMD_START                        // 立即发送
                    | 24;                                   // SD 命令 24，CMD24
while((*P_SD_INT_STATUS & C_SD_CMDBUF_FULL) == 0);          // 等待响应寄存器满
response = *P_SD_RESPONSE_DATA;                             // 保存响应数据
while((*P_SD_INT_STATUS & C_SD_CMD_COMPLETE) == 0);         // 等待命令传输完毕
for(i=0; i<512/4; i++)                                      // 发送一个扇区 512 字节数据
{
    while((*P_SD_INT_STATUS & C_SD_DATABUF_EMPTY) == 0);// 等待发送缓冲区空
    *P_SD_RX_DATA = BlockData[i];                           // 以字为单位发送的数据
}
while((*P_SD_INT_STATUS & C_SD_DATA_COMPLETE) == 0); // 等待数据传送完毕
```

4.15 TFT LCD 控制器

4.15.1 概述

TFT LCD 发展现状

LCD 是 Liquid Crystal Display 的缩写，是一种显示器件。当前 LCD 种类众多 TN（Twisted Nematic）、STN（Super TN）、DSTN（Double STN）、CSTN（Color STN）、FSTN（Film STN）、UFB（CSTN）及 TFT LCD。当前 LCD 发展迅速，应用在各种显示场合，尤其以 TFT-LCD 发展更为迅猛。

TFT LCD 是 Thin Film Transistor-Liquid Crystal Display 的缩写，即薄膜晶体管液晶显示器。在驱动方式上，TFT LCD 与无源 TN-LCD、STN-LCD 的简单矩阵不同，它在液晶显示屏的每一个象素上都设置有一个薄膜晶体管（TFT），可有效地克服非选通时的串扰，使显示液晶屏的静态特性与扫描线数无关，大大提高了图像质量（尺寸、色彩），控制起来也比较容易。

当前生产 TFT-LCD 的主要厂商有日本的夏普、韩国的三星、LG、台湾的友达光电、奇美电子、中华映管、翰宇彩晶、广辉电子等。

TFT LCD 的结构

TFT LCD 的面板排布如图 4.92 所示：

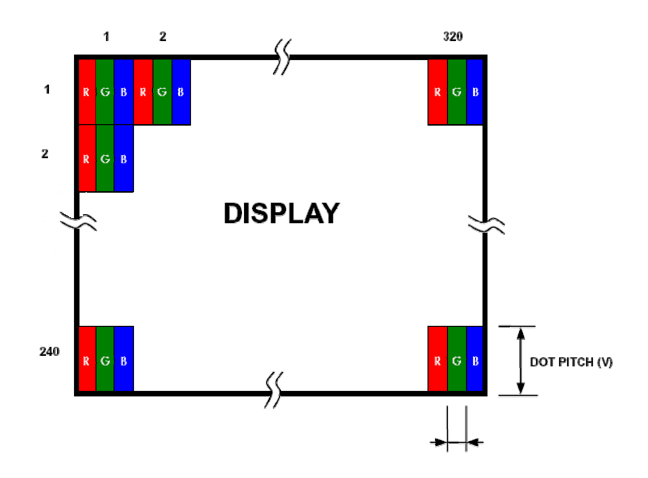


图 4.92 TFT LCD 的面板排布

TFT LCD 的结构如图 4.93 所示：主要由偏振片、滤色器基板、液晶、TFT 基板、偏振片、背光源组成。

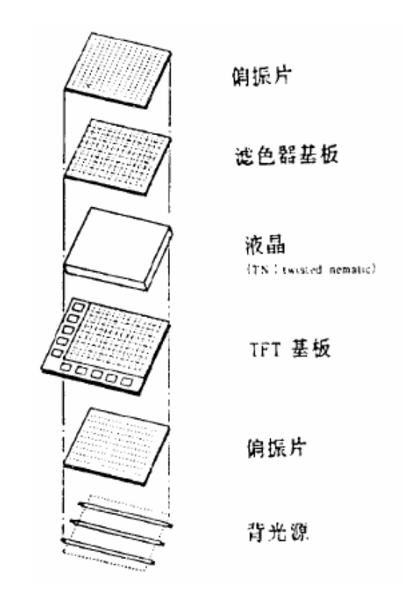


图 4.93 TFT LCD 的结构

一个 TFT 单元的结构如图 4.94 所示：

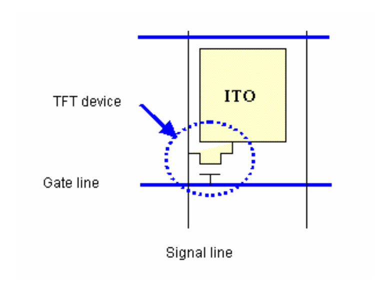


图 4.94 TFT 的一个控制单元

TFT LCD 的显示原理

通过改变 R、G、B 信号线上的电压，可以控制 LCD 的透光，使每一个像素呈现不同的颜色。按照时序，控制整屏的所有像素点就可以控制 LCD 显示。

内置于 SPCE3200 的 TFT LCD 接口能够支持对多种输入格式的 TFT 型 LCD 显示屏的控制，譬如 DataEnable（DE）、Hsync/Vsync、16 位并行 RGB、8 位 delta RGB 以及 CCIR601/656 等输入。LCD 显示屏支持的分辨率为 320（水平像素）× 240（垂直像素），并支持 NTSC/PAL 显示格式。由于同步信号的位置及宽度都是可配置的，故可以满足各种规格的 TFT LCD 屏的要求。

4.15.2 特性

SPCE3200 的 TFT-LCD 控制器支持下列数据格式：并行 RGB 格式、串行 RGB 格式、串行 RGBDm 格式、CCIR-601/CCIR-656 格式。支持多种时钟频率，系统时钟（27MHz）、系统时钟/2（13.5MHz）、系统时钟/4（6.75MHz）、系统时钟/8（3.375MHz）。

4.15.3 引脚描述

SPCE3200 的 TFT 控制器相关 I/O 管脚：LCD_CLK（时钟信号）、LCD_ACT（数据使能信号）、LCD_VS（垂直同步信号）、LCD_HS（水平同步信号）、LCD_Data[15:0]（数据总线），具体参考下表：

表 4.174 TFT 管脚描述表

引脚名称	引脚号	引脚属性	引脚功能
LCD_CLK	148	O	LCD 的时钟信号
LCD_VS	147	O	LCD 的垂直输出信号
LCD_HS	146	O	LCD 的水平输出信号
LCD_D0	144	O	LCD 的数据输出总线
LCD_D1	143	O	
LCD_D2	142	O	
LCD_D3	141	O	
LCD_D4	140	O	
LCD_D5	139	O	
LCD_D6	138	O	
LCD_D7	137	O	
LCD_D8	136	O	
LCD_D9	135	O	
LCD_D10	134	O	
LCD_D11	133	O	
LCD_D12	132	O	
LCD_D13	131	O	
LCD_D14	130	O	
LCD_D15	129	O	

LCD_ACT	145	O	LCD 的数据使能信号

4.15.4 寄存器描述

TFT LCD 控制器共有 32 个寄存器，如表 4.175 所示：通过对这 32 个寄存器的操作，即可通过 TFT LCD 控制器控制显示。

表 4.175 TFT 控制器相关寄存器列表

寄存器名称	助记符	地址
TFT 控制寄存器	P_TFT_MODE_CTRL	0x88040000
TFT 输出数据格式寄存器	P_TFT_DATA_FMT	0x88040004
TFT 水平显示扫描寄存器	P_TFT_HOR_ACT	0x88040008
TFT 水平空白前扫寄存器	P_TFT_HOR_FRONT	0x8804000C
TFT 水平空白后扫寄存器	P_TFT_HOR_BACK	0x88040010
TFT 水平同步扫描寄存器	P_TFT_HOR_SYNC	0x88040014
TFT 垂直显示扫描寄存器	P_TFT_VER_ACT	0x88040018
TFT 垂直空白前扫寄存器	P_TFT_VER_FRONT	0x8804001C
TFT 垂直空白后扫寄存器	P_TFT_VER_BACK	0x88040020
TFT 垂直同步扫描寄存器	P_TFT_VER_SYNC	0x88040024
TFT 帧数据格式寄存器 1	P_TFT_FRAME_FMT1	0x88040028
TFT 起始行寄存器	P_TFT_ROW_START	0x8804002C
TFT 起始列寄存器	P_TFT_COL_START	0x88040030
TFT 列宽度寄存器	P_TFT_COL_WIDTH	0x88040034
TFT 余量宽度寄存器	P_TFT_DUMMY_WIDTH	0x88040038
TFT 状态寄存器	P_TFT_BUFFER_STATUS	0x8804003C
TFT 输出数据顺序寄存器	P_TFT_DATA_SEQ	0x88040040
TFT 中断状态寄存器	P_TFT_INT_STATUS	0x88040050
TFT 帧数据格式寄存器 2	P_TFT_FRAME_FMT2	0x880400A0
LCD 时钟配置寄存器	P_LCD_CLK_CONF	0x88210034
LCD 时钟选择寄存器	P_LCD_CLK_SEL	0x88210038
LCD 接口选择寄存器	P_LCD_INTERFACE_SEL	0x88200000
LCD 显示缓冲区起始地址寄存器 1	P_LCD_BUFFER_SA1	0x8807000C
LCD 显示缓冲区起始地址寄存器 2	P_LCD_BUFFER_SA2	0x88070010

LCD 显示缓冲区起始地址寄存器 3	P_LCD_BUFFER_SA3	0x88070014
LCD 显示缓冲区选择寄存器	P_LCD_BUFFER_SEL	0x88090024
TFT GPIO 输出数据寄存器	P_TFT_GPIO_DATA	0x88200014
TFT GPIO 输出使能寄存器	P_TFT_GPIO_OUTPUTEN	0x88200018
TFT GPIO 上拉寄存器	P_TFT_GPIO_PULLUP	0x8820001C
TFT GPIO 下拉寄存器	P_TFT_GPIO_PULLDOWN	0x88200020
TFT GPIO 输入数据寄存器	P_TFT_GPIO_INPUT	0x88200064
TFT GPIO 外部中断寄存器	P_TFT_GPIO_INT	0x88200080

如果使用 TFT LCD 接口复用为 GPIO 功能，可以对下面的寄存器操作。寄存器的各位对应引脚关系如表 4.176：

表 4.176 TFT 接口复用为 GPIO 引脚与寄存器位对应关系

引脚名称	引脚号	控制寄存器位	是否可以作为外部中断
LCD_CLK	148	bit[0]	可以
LCD_VS	147	bit[1]	可以
LCD_HS	146	bit[2]	可以
LCD_D0	144	bit[3]	可以
LCD_D1	143	bit[4]	不可以
LCD_D2	142	bit[5]	不可以
LCD_D3	141	bit[6]	不可以
LCD_D4	140	bit[7]	不可以
LCD_D5	139	bit[8]	不可以
LCD_D6	138	bit[9]	不可以
LCD_D7	137	bit[10]	不可以
LCD_D8	136	bit[11]	不可以
LCD_D9	135	bit[12]	不可以
LCD_D10	134	bit[13]	不可以
LCD_D11	133	bit[14]	不可以
LCD_D12	132	bit[15]	不可以
LCD_D13	131	bit[16]	不可以
LCD_D14	130	bit[17]	不可以



LCD_D15	129	bit[18]	不可以
LCD_ACT	145	bit[19]	不可以

TFT GPIO 输出数据寄存器：P_TFT_GPIO_DATA(0x88200014)

TFT GPIO 输出数据寄存器设置 TFT 控制器接口复用为 GPIO 时的输出数据。

表 4.177 P_TFT_GPIO_DATA(0x88200014)

位	b31~b20	b19~b0
读/写	-	R/W
默认值	-	0
名称	-	TFT_GPIO_O

TFT_GPIO_O b19~b0 TFT_GPIO_O：TFT 接口作为 GPIO 使用时的输出数据

TFT GPIO 输出使能寄存器：P_TFT_GPIO_OUTPUTEN(0x88200018)

TFT GPIO 输出使能寄存器设置 TFT 接口复用为 GPIO 时引脚输出使能。

表 4.178 P_TFT_GPIO_OUTPUTEN(0x88200018)

位	b31~b20	b19~b0
读/写	-	R/W
默认值	-	0
名称	-	TFT_GPIO_OE

TFT_GPIO_OE b19~b0 TFT_GPIO_OE：TFT 接口作为 GPIO 使用时的输出使能位：

0：不使能输出

1：使能输出

TFT GPIO 上拉寄存器：P_TFT_GPIO_PULLUP(0x8820001C)

TFT GPIO 上拉寄存器设置 TFT 接口复用为 GPIO 时的上拉电阻输入使能。

表 4.179 P_TFT_GPIO_PULLUP(0x8820001C)

位	b31~b20	b19~b0
读/写	-	R/W
默认值	-	0
名称	-	TFT_GPIO_PU

TFT_GPIO_PU　b19~b0　　TFT_GPIO_PU：TFT 接口作为 GPIO 使用时的上拉电阻输入使能位：

0：使能上拉电阻输入

1：不使能上拉电阻输入

TFT GPIO 下拉寄存器：P_TFT_GPIO_PULLDOWN(0x88200020)

TFT GPIO 下拉寄存器设置 TFT 接口复用为 GPIO 时的下拉电阻输入使能。

表 4.180 P_TFT_GPIO_PULLDOWN(0x88200020)

位	b31~b20	b19~b0
读/写	-	R/W
默认值	-	0
名称	-	TFT_GPIO_PD

TFT_GPIO_PD　b19~b0　　TFT_GPIO_PD：TFT 接口作为 GPIO 使用时的下拉电阻输入使能位：

0：不使能下拉电阻输入

1：使能下拉电阻输入

TFT GPIO 输入数据寄存器：P_TFT_GPIO_INPUT(0x88200064)

TFT GPIO 输入数据寄存器保存 TFT 接口复用为 GPIO 时的外部输入数据。

表 4.181 P_TFT_GPIO_INPUT(0x88200064)

位	b31~b20	b19~b0
读/写	-	R/W
默认值	-	0
名称	-	TFT_INPUT

TFT_INPUT　b19~b0　　TFT 接口作为 GPIO 使用时的输入数据

TFT GPIO 外部中断寄存器：P_TFT_GPIO_INT(0x88200080)

TFT GPIO 外部中断寄存器可进行中断使能、中断触发沿设置、中断标志清除等操作。

表 4.182 P_TFT_GPIO_INT(0x88200080)

位	b31~b28	b27~b24	b23~b20	b19~b16	b15~b12	b11~b8	b7~b4	b3~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W
默认值	-	0	-	0	-	0	-	0



名称	-	TFT_FI	-	TFT_RI	-	TFT_FIEN	-	TFT_RIEN

TFT_FI　b27~b24　TFT_FI：TFT 接口复用为 GPIO 下降沿中断标志位：

读 0：没有发生下降沿中断

读 1：发生下降沿中断

写 0：无意义

写 1：清除中断标志

TFT_RI　b19~b16　TFT_RI：TFT 接口复用为 GPIO 上升沿中断标志位：

读 0：没有发生上升沿中断

读 1：发生上升沿中断

写 0：无意义

写 1：清除中断标志

TFT_FIEN　b11~b8　TFT_FIEN：TFT 接口复用为 GPIO 下降沿中断使能位：

0：不使能下降沿中断

1：使能下降沿中断

TFT_RIEN　b3~b0　TFT_RIEN：TFT 接口复用为 GPIO 上升沿中断使能位：

0：不使能上升沿中断

1：使能上升沿中断

作为 TFT 控制器接口需要设置以下寄存器：

LCD 时钟配置寄存器：P_LCD_CLK_CONF(0x88210034)

LCD 时钟配置寄存器用于使能或禁止 TFT 控制器模块时钟，或软复位 TFT 控制器模块。

表 4.183 P_LCD_CLK_CONF(0x88210034)

位	b31~b2	b1	b0
读/写	-	R/W	R/W
默认值	-	1	0
名称	-	LCD_RST	LCD_STOP

LCD_RST　b1　LCD 模块时钟复位位：

0：LCD 模块时钟复位

1：LCD 模块时钟不复位

LCD_STOP　b0　LCD 模块时钟使能位：

0：LCD 模块时钟停止

1：LCD 模块时钟使能

LCD 时钟选择寄存器：P_LCD_CLK_SEL(0x88210038)

为满足不同 TFT LCD 的时钟频率，SPCE3200 控制器提供用户选择配置时钟频率，TFT LCD 默认时钟使用 APB 总线时钟 27MHz。如果用户选择 PLLU 时钟作为频率源，需要先通过 P_CLK_PLLAU_CONF 寄存器使能 PLLU。

表 4.184 P_LCD_CLK_SEL(0x88210038)

位	b31~b6	b5~b3	b2~b0
读/写	-	R/W	R/W
默认值	-	0	0
名称	-	TFT_CLK	STN_CLK

TFT_CLK　b5~b3　选择 TFT LCD 时钟频率：

000：27MHz

001：USB PLL 输出频率 2 分频

010：USB PLL 输出频率 3 分频

011：USB PLL 输出频率 4 分频

100：USB PLL 输出频率 6 分频

101：USB PLL 输出频率 8 分频

STN_CLK　b2~b0　选择 STN LCD 时钟频率：

000：视频 PLL 输出频率

001：视频 PLL 输出频率 2 分频

010：视频 PLL 输出频率 3 分频

011：视频 PLL 输出频率 4 分频

100：视频 PLL 输出频率 6 分频

101：视频 PLL 输出频率 8 分频

LCD 接口选择寄存器：P_LCD_INTERFACE_SEL(0x88200000)

LCD 接口选择寄存器设置控制器作为 TFT 接口、STN 接口等。

表 4.185 P_LCD_INTERFACE_SEL(0x88200000)

位	b31~b2	b1~b0
读/写	-	R/W
默认值	-	0
名称	-	SW_LCD

SW_LCD　　b1~b0　　SW_LCD：LCD 端口配置

00：不做为 LCD 接口

01：TFT_AUO 接口

10：TFT_TOPPOLY 接口

11：STN 接口

TFT 控制寄存器：P_TFT_MODE_CTRL(0x88040000)

TFT 控制寄存器选择 TFT 输出时钟、图像水平、垂直的缩放比例、使能 TFT LCD 控制器等。

表 4.186 P_TFT_MODE_CTRL(0x88040000)

位	b31~b25	b24	b23~b12	b11~b10	b9~b8	b7~b2	b1~b0
读/写	-	R/W	-	R/W	R/W	-	R/W
默认值	-	0	-	0	0	-	0
名称	-	TFT_EN	-	VER_SCALING	HOR_SCALING	-	TFT_CLK_SEL

TFT_EN　　b24　　TFT_EN：TFT-LCD 使能位：

0：不使能

1：使能

VER_SCALING　　b11~b10　　VER_SCALING：图像垂直缩放位：

00：不缩放

01：显示图像垂直方向放大一倍

10：显示图像垂直方向缩小一倍

11：不缩放

HOR_SCALING　　b9~b8　　HOR_SCALING：图像水平缩放位：

00：不缩放

01：显示图像水平方向放大一倍

10：显示图像水平方向缩小一倍

11：不缩放

TFT_CLK_SEL　　b1~b0　　TFT_CLK_SEL：TFT-LCD 输出时钟选择位：

0：系统时钟（27MHz）

01：系统时钟/2（13.5MHz）

10：系统时钟/4（6.75MHz）

11：系统时钟/8（3.875MHz）

TFT 中断状态寄存器：P_TFT_INT_STATUS(0x88040050)

TFT 中断状态寄存器控制垂直空白中断允许及清除垂直空白中断标志。

表 4.187 P_TFT_INT_STATUS(0x88040050)

位	b31~b25	b24	b23~b17	b16	b15~b0
读/写	-	R/W	-	R/W	-
默认值	-	0	-	0	-
名称	-	INT_EN	-	VBLK_INT	-

INT_EN　　b24　　INT_EN：垂直空白中断使能位：

0：不使能

1：使能

VBLK_INT　　b16　　VBLK_INT：垂直空白中断标志位：

读 0：没有发生垂直空白中断

读 1：发生垂直空白中断

写 0：无意义

写 1：清除中断标志

TFT 状态寄存器：P_TFT_BUFFER_STATUS(0x8804003C)

TFT 状态寄存器反映 TFT 缓存区与 TFT 控制器的状态。

表 4.188 P_TFT_BUFFER_STATUS(0x8804003C)

位	b31~b25	b24	b23~b17	b16	b15~b0
读/写	-	R/W	-	R/W	-
默认值	-	0	-	0	-
名称	-	BUF_ERR	-	TFT_FINISH	-

BUF_ERR　　b24　　BUF_ERR：TFT-LCD 缓存区下溢标志，当 LCD 缓存区的读取速度快于写入速度时，TFT-LCD 控制器会将这一位置 1

读 0：未发生下溢

读 1：发生下溢

写 0：没有意义

写 1：清除标志

TFT_FINISH　　b16　　TFT_FINISH：这一位置 1 表示 TFT 控制器完成了内部操作，用户此时可以关闭 TFT 时钟

TFT 显示缓存区相关寄存器：

LCD 显示缓冲区起始地址寄存器 1：P_LCD_BUFFER_SA1(0x8807000C)



LCD 显示缓冲区起始地址寄存器 2：P_LCD_BUFFER_SA2(0x88070010)

LCD 显示缓冲区起始地址寄存器 3：P_LCD_BUFFER_SA3(0x88070014)

这 3 个寄存器将存储 LCD 显示缓冲区的起始地址，注意起始地址的低 8 位必须为 0。

表 4.189 P_LCD_BUFFER_SA1/2/3

位	b31~b24	b23~b0
读/写	-	R/W
默认值	-	0
名称	-	PPU_LCD_ADR1/2/3

PPU_LCD_ADRx	b23~b0	LCD 帧缓冲区 x（1、2、3）的起始地址 b7~b0 必须为 0

LCD 显示缓冲区选择寄存器：P_LCD_BUFFER_SEL(0x88090024)

LCD 显示缓冲区选择寄存器选择当前具体使用哪一个帧缓冲区。

表 4.190 P_LCD_BUFFER_SEL(0x88090024)

位	b31~b2	b1~b0
读/写	-	R/W
默认值	-	0
名称	-	LCD_PTR

LCD_PTR	b1~b0	LCD_PTR：写入该寄存器是将当前帧缓存区指向 LCD 帧缓存区 0~2 中之一；读出该寄存器是为了获知当前帧缓存区指向 LCD 帧缓存区 0~2 中哪一个 0：LCD frame buffer0 1：LCD frame buffer1 2：LCD frame buffer2

TFT 数据格式相关寄存器：

TFT 输出数据格式寄存器：P_TFT_DATA_FMT(0x88040004)

TFT 输出数据格式寄存器选择输出数据格式是并行 RGB 格式、串行 RGB 格式、串行 RGBDm 格式、CCIR601 格式还是 CCIR656 格式，在选择 CCIR656 格式后，选择输出有效数据是 640 个象素点，还是 720 个象素点。

表 4.191 P_TFT_DATA_FMT(0x88040004)

位	b31~b9	b8	b7~b3	b2~b0

读/写	-	R/W	-	R/W
默认值	-	0	-	0
名称	-	CCIR656_M	-	TFT_FMT

CCIR656_M　　b8　　CCIR656_M：标准 CCIR656 格式

0：控制器会输出 640 个有效像素的显示

1：控制器会输出 720 个有效像素的显示，但在图像两端有 80 个像素的黑屏

TFT_FMT　　b2~b0　　TFT_FMT：TFT-LCD 输出数据格式

000：并行 RGB 格式

001：串行 RGB 格式

010：串行 RGBDm 格式

011：CCIR601 格式

100：CCIR656 格式

TFT 帧数据格式寄存器 1：P_TFT_FRAME_FMT1(0x88040028)

TFT 帧数据格式寄存器选择帧缓冲区数据格式。

表 4.192 P_TFT_FRAME_FMT1(0x88040028)

位	b31~b2	b1~b0
读/写	-	R/W
默认值	-	0
名称	-	FB_FMT

FB_FMT　　b1~b0　　FB_FMT：帧缓冲区数据格式

00：RGB565 格式（小端模式）

01：RGB1555 格式（大端模式）

10：YUYV 格式（小端模式）

11：4Y4U4Y4V 格式（小端模式）

TFT 输出数据顺序寄存器：P_TFT_DATA_SEQ(0x88040040)

TFT 输出数据顺序寄存器选择对外输出数据的顺序。

表 4.193 P_TFT_DATA_SEQ(0x88040040)

位	b31~b25	b24	b23~b19	b18~b16	b15~b11	b10~b8	b7~b3	b2~b0
读/写	-	R/W	-	R/W	-	R/W	-	R/W

默认值	-	0	-	0	-	0	-	0
名称	-	YUV_FMT	-	YUV_SEQ	-	RGB_OLSEQ	-	RGB_ELSEQ

YUV_FMT　b24　YUV_FMT：输出数据格式选择

0：YCbCr 格式

1：YUV 格式

YUV_SEQ　b18~b16　YUV_SEQ：YUV 数据输出序列选择(针对 CCIR601 及 CCIR656 输入格式）

000：Y0U0Y1V0/Y0Cb0Y1Cr0

001：Y0V0Y1U0/Y0Cr0Y1Cb0

010：U0Y0V0Y1/Cb0Y0Cr0Y1

011：V0Y0U0Y1/Cr0Y0Cb0Y1

100：Y1U0Y0V0/Y1Cb0Y0Cr0（为测试保留）

101：Y1V0Y0U0/Y1Cr0Y0Cb0（为测试保留）

110：U0Y1V0Y0/Cb0Y1Cr0Y0（为测试保留）

111：V0Y1U0Y0/Cr0Y1Cb0Y0（为测试保留）

RGB_OLSEQ　b10~b8　RGB_OLSEQ：RGB 数据输出序列（针对串行 RGB 和串行 RGBDm 输入数据的奇数行）

000：RGB（RGBDm）

001：GBR（GBRDm）

010：BRG（BRGDm）

011：RBG（RBGDm）

100：BGR（BGRDm）

101：GRB（GRBDm）

RGB_ELSEQ　b2~b0　RGB_ELSEQ：RGB 数据输出序列（针对串行 RGB 和串行 RGBDm 输入数据的偶数行）

000：RGB（RGBDm）

001：GBR（GBRDm）

010：BRG（BRGDm）

011：RBG（RBGDm）

100：BGR（BGRDm）

101：GRB（GRBDm）

TFT 帧数据格式寄存器 2：P_TFT_FRAME_FMT2(0x880400A0)

TFT 帧数据格式寄存器选择帧缓冲区数据格式。

表 4.194 P_TFT_FRAME_FMT2(0x880400A0)

位	b31~b1	b0
读/写	-	R/W
默认值	-	0
名称	-	FB_YCbCr

FB_YCbCr　　b0　　FB_YCbCr：帧缓存区数据格式选择

0：YUV 格式

1：YCbCr 格式

TFT 行设置相关寄存器：

TFT 水平显示扫描寄存器：P_TFT_HOR_ACT(0x88040008)

根据 TFT LCD 显示器的数据手册，填写水平显示时间。

表 4.195 P_TFT_HOR_ACT(0x88040008)

位	b31~b12	b11~b0
读/写	-	R/W
默认值	-	0
名称	-	HOR_ACT

HOR_ACT　　b11~b0　　HOR_ACT：水平活动区

单位：系统时钟周期，注意时钟周期是指 27MHz

TFT 水平空白前扫寄存器：P_TFT_HOR_FRONT(0x8804000C)

根据 TFT LCD 显示器的数据手册，填写水平空白前扫时间。

表 4.196 P_TFT_HOR_FRONT(0x8804000C)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	HOR_FBLK

HOR_FBLK　　b9~b0　　HOR_FBLK：前水平空白区

单位：系统时钟周期

TFT 水平空白后扫寄存器：P_TFT_HOR_BACK(0x88040010)



根据 TFT LCD 显示器的数据手册，填写水平空白后扫时间。

表 4.197 P_TFT_HOR_BACK(0x88040010)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	HOR_BBLK

HOR_BBLK　　b9~b0　　HOR_BBLK：后水平空白区
单位：系统时钟周期

TFT 水平同步扫描寄存器：P_TFT_HOR_SYNC(0x88040014)

根据 TFT LCD 显示器的数据手册，填写水平同步扫描时间。另外该寄存器还可以选择水平同步脉冲的极性。

表 4.198 P_TFT_HOR_SYNC(0x88040014)

位	b31~b25	b24	b23~b8	b7~b0
读/写	-	R/W	R/W	R/W
默认值	-	0	0	0
名称	-	HS_P	-	HOR_SYNCW

HS_P　　b24　　HS_P：水平同步脉冲的极性
0：负极性
1：正极性

HOR_SYNCW　　b7~b0　　HOR_SYNCW：水平同步脉冲宽度
单位：系统时钟周期

TFT 场设置相关寄存器：

TFT 垂直显示扫描寄存器：P_TFT_VER_ACT(0x88040018)

根据 TFT LCD 显示器的数据手册，填写垂直显示扫描时间。

表 4.199 P_TFT_VER_ACT(0x88040018)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	VER_ACT

VER_ACT　　b9~b0　　VER_ACT：垂直活动
单位：行

TFT 垂直空白前扫寄存器：P_TFT_VER_FRONT(0x8804001C)

根据 TFT LCD 显示器的数据手册，填写垂直空白前扫时间。

表 4.200 P_TFT_VER_FRONT(0x8804001C)

位	b31~b8	b7~b0
读/写	-	R/W
默认值	-	0
名称	-	VER_FBLK

VER_FBLK　　b7~b0　　VER_FBLK：前垂直空白区
单位：行

TFT 垂直空白后扫寄存器：P_TFT_VER_BACK(0x88040020)

根据 TFT LCD 显示器的数据手册，填写垂直空白后扫时间。

表 4.201 P_TFT_VER_BACK(0x88040020)

位	b31~b8	b7~b0
读/写	-	R/W
默认值	-	0
名称	-	VER_BBLK

VER_BBLK　　b7~b0　　VER_BBLK：后垂直空白区
单位：行

TFT 垂直同步扫描寄存器：P_TFT_VER_SYNC(0x88040024)

根据 TFT LCD 显示器的数据手册，填写垂直同步扫描时间。另外该寄存器还可以选择垂直同步脉冲的极性。

表 4.202 P_TFT_VER_SYNC(0x88040024)

位	b31~b25	b24	b23~b5	b4~b0
读/写	-	R/W	R/W	R/W
默认值	-	0	0	0
名称	-	VS_P	-	VER_SYNCW

VS_P　　　　b24　　　　VS_P：垂直同步脉冲的极性
　　　　　　　　　　　　　0：负极性
　　　　　　　　　　　　　1：正极性
VER_SYNCW　b4~b0　　　VER_SYNCW：垂直同步脉冲宽度
　　　　　　　　　　　　　设定范围：1~31

TFT 显示位置相关寄存器：

TFT 起始行寄存器：P_TFT_ROW_START(0x8804002C)

TFT 起始行寄存器设置起始行开始位置，参考图 4.95。

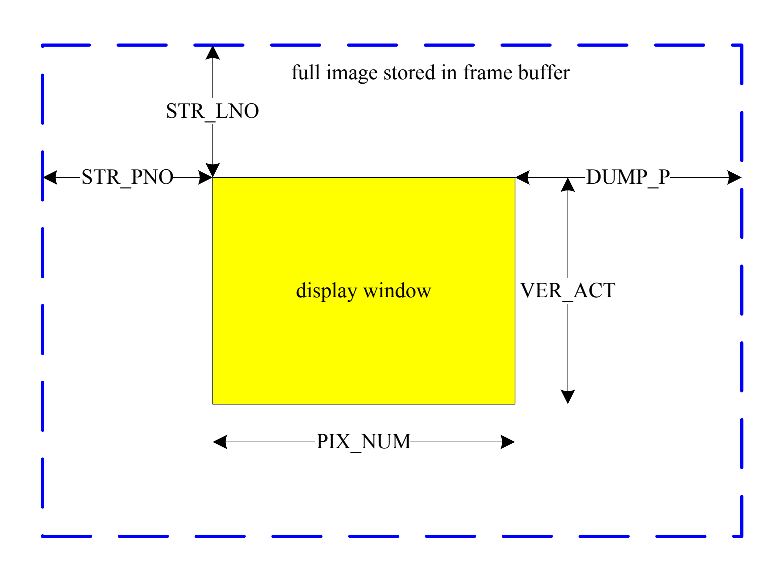


图 4.95 TFT LCD 显示结构图

表 4.203 P_TFT_ROW_START(0x8804002C)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	STR_LNO

STR_LNO　　b9~b0　　　STN_LNO：帧缓冲区起始行的行数
　　　　　　　　　　　　　单位：行

TFT 起始列寄存器：P_TFT_COL_SRART(0x88040030)

TFT 起始列寄存器设置起始列开始位置，参考图 4.95。

表 4.204 P_TFT_COL_START(0x88040030)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	STR_PNO

STR_PNO　　b9~b0　　STN_PNO：帧缓冲区起始像素数，须为 16 的整数倍
单位：像素

TFT 列宽度寄存器：P_TFT_COL_WIDTH(0x88040034)

TFT 列宽度寄存器设置 TFT 屏幕列显示宽度，参考图 4.95。

表 4.205 P_TFT_COL_WIDTH(0x88040034)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	PIX_NUM

PIX_NUM　　b9~b0　　PIX_NUM：帧缓存区内一行的图像像素数，须为 16 的整数倍
单位：像素（16 位）

TFT 余量宽度寄存器：P_TFT_DUMMY_WIDTH(0x88040038)

TFT 余量宽度寄存器设置 TFT 显示屏幕与帧缓冲区之间宽度。

表 4.206 P_TFT_DUMMY_WIDTH(0x88040038)

位	b31~b10	b9~b0
读/写	-	R/W
默认值	-	0
名称	-	DUMP_PIX

DUMP_PIX　　b9~b0　　DUMP_PIX：帧缓存区内一行的图像像素数，须为 16 的整数倍
单位：像素（16 位）

4.15.5 基本操作

使用 SPCE3200 TFT 控制器点亮台盛公司的 TS35ND5B01 屏，完成初始化程序。

```
#include "SPCE3200_Register.h"
```

```
#include "SPCE3200_Constant.h"

void LCD_Buffer_Setting(int FB_ADDR0, int FB_ADDR1, int FB_ADDR2)
{
    *P_LCD_BUFFER_SA1 = FB_ADDR0;
    *P_LCD_BUFFER_SA2 = FB_ADDR1;
    *P_LCD_BUFFER_SA3 = FB_ADDR2;
}

void Init_TFTLcd(void)
{

    *P_LCD_CLK_CONF = C_LCD_RST_DIS
                    | C_LCD_CLK_EN;                  // LCD 模式时钟使能
    *P_LCD_INTERFACE_SEL = C_LCD_PORT_AUO;           // 管脚复用，选择 TFT_AUO 模式

    // 设置 TFT-LCD 数据模式
    *P_TFT_FRAME_FMT1 = C_TFT_BUF_4Y4U4Y4V;          // 数据帧格式选择 4Y4U4Y4V
    *P_TFT_DATA_FMT = C_TFT_PARALLEL_RGB;            // TFT-LCD 管脚输出数据格式为并
行 RGB 格式
    *P_TFT_DATA_SEQ = C_TFT_OUTPUT_YCBCR;            // TFT-LCD 输出数据格式为 YCbCr
    *P_TFT_FRAME_FMT2 = C_TFT_BUF_YCBCR;             // TFT-LCD 帧数据格式为 YCbCr

    // 设置 TFT-LCD 的行信号
    *P_TFT_HOR_ACT = 1280;
    *P_TFT_HOR_FRONT = 80;
    *P_TFT_HOR_BACK = 152;
    *P_TFT_HOR_SYNC = 120;

    // 设置 TFT-LCD 的场信号
    *P_TFT_VER_ACT = 240;
    *P_TFT_VER_FRONT = 4;
    *P_TFT_VER_BACK = 15;
    *P_TFT_VER_SYNC = 3;

    // 设置 TFT-LCD 显示的起始位置
```

```
        *P_TFT_ROW_START = 0;
        *P_TFT_COL_START = 0;
        *P_TFT_COL_WIDTH = 320;
        *P_TFT_DUMMY_WIDTH = 0;

        // 设置 TFT-LCD 的控制寄存器
        *P_TFT_INT_SEL = ~C_TFT_INT_EN;                    // 中断不使能
        // 使能 TFT 模块，图像不放大，选择 6.75MHz 时钟
        *P_TFT_CTRL_SEL = C_TFT_CTRL_EN | C_TFT_CLK_27MDIV4;
}
```



5 SPCE3200 开发系统介绍（开发板及开发工具）

通常利用 SPCE3200 芯片进行开发时，硬件上需要一套 SPCE3200 开发板作为开发平台，需要 PC 机进行软件程序开发，同时需要一套下载调试程序的工具（本书中特指 SJPROBE）把 PC 机上编写好的程序下载到开发板上；软件上需要凌阳科技 S+core 集成开发环境 S+core IDE 支持程序的开发。

5.1 开发板

SPCE3200 开发板是凌阳科技大学计划推出的 32 位嵌入式开发板，是为大专院校作为嵌入式操作系统、嵌入式硬件设计教学设备，供学生学习和研究量身打造的，同时也适合研究开发人员和电子爱好者进行评估和开发。该开发板设计基于开发考虑，配置简单，布局简洁明了，外设齐全，接口标准，扩展方便。采用四层板设计，线路稳定。SPCE3200 芯片是开发板上的核心器件，采用凌阳自主设计的 S+core 内核，并且集成了 MPEG4 的硬件编解码以及 CMOS 传感器 / TV 解码接口，低功耗，高性能，适用于 PDA，便携媒体播放器，机器人等设备与终端。

5.1.1 功能特点

SPCE3200 开发板具有板上资源丰富、接口方便等特点，具体如下：

- 开发板以 SPCE3200 芯片为核心，SPCE3200 采用 Sunplus S^+core®处理器，频率高达 162MHz
- 提供 DRAM 接口：可以通过 DRAM 接口访问外部 ROM 或 NOR Flash，来实现 boot 程序或应用程序的存储
- RAM：128M bit SDRAM
- 64M bit NOR FLASH
- 提供 Nand Flash 接口：支持 64M Byte Nand Flash
- 10M 以太网接口。采用 10Mbps 低功耗嵌入式专用以太网芯片 ENC28J60，接口为标准 RJ45 插座，集成网络变压器，安全可靠
- UART 接口及 JOY STICK 接口。
- USB 1.1 主从通讯接口。通过跳线选择 USB 接口作为主机接口或者设备接口
- SD 卡接口
- I^2C 接口。支持 I^2C（主模式），有 3 种传输类型：8 位、16 位和 8 位×n
- SPI 主从通讯接口
- SIO 接口
- LCD 接口。支持多种 TFT LCD 的输入格式，该接口支持 320(H) × 240(V) 的解析度以及 NTSC/PAL 制式；同步信号的宽度及位置是可以配置的，以适应不同规格的 TFT LCD 板。支持 STN LCD
- 四线触摸屏接口
- 音视频接口：即支持标准 TV 接口，也支持类似耳机、音响等设备的输出
- 硬件 MPEG-4/JPEG 编解码。MPEG-4 帧率(frame rate)：CIF 模式下，高达 30 帧/秒
- CSI I/F 摄像头接口（凌阳 CMOS 图像传感器接口），支持 CCIR-601/656 CMOS 图像传感器
- 提供通用 A/D 接口和 MIC 输入和 Line in 输入接口：即支持开发板上提供的 MIC 输入语音，

也支持具有 Line in 接口的 MIC 输入语音

- GPS/GPRS 接口
- IPOD KEY 接口
- SJTAG 接口，可直接与凌阳科技提供的 SJ Probe 连接进行程序烧录和在线调试
- 三个按键
- 三个 LED

5.1.2 硬件原理

硬件布局

SPCE3200 开发板的布局结构如图 5.1：在开发板上，SPCE3200 及其周边电路分布在开发板的中央，作为开发板的控制核心；开发板的左端为电源电路，为整个开发板提供工作电压；在 SPCE3200 的四周，分布了 SDRAM、Nor Flash、Nand Flash 等，作为 SPCE3200 的片外存储设备，开发板上有两片 64M bit 的 SDRAM 并联，构成 128M bit；在开发板的四个边上，分布了包含以太网、UART、USB、SD 卡、音视频等接口，可以方便地与外围各功能设备连接。

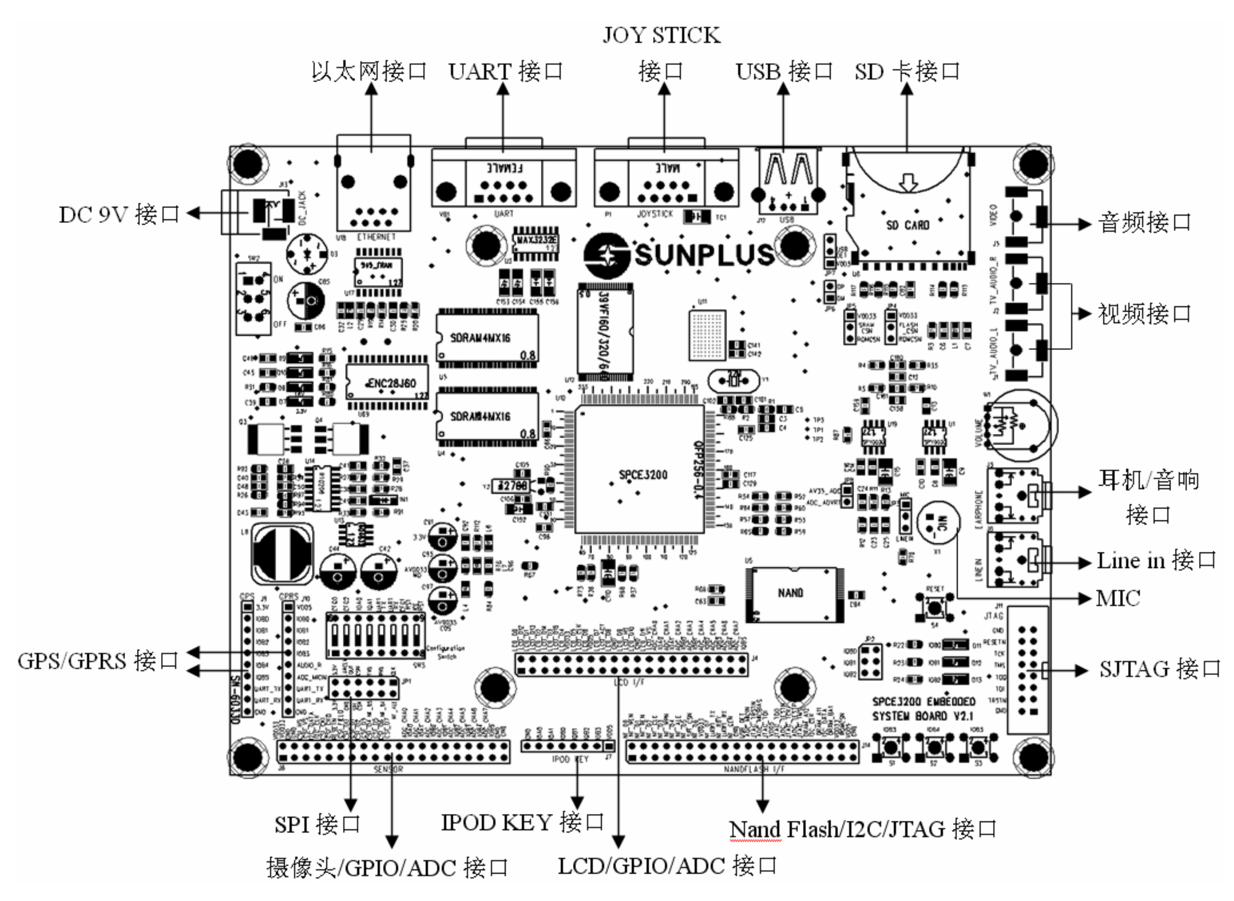


图 5.1 SPCE3200 开发板布局结构图

为了使得 SPCE3200 开发板各模块能更好地协调工作，在开发板上设计了 8 组跳线和一个配置开关，如表 5.1。

表 5.1 SPCE3200 开发板跳线及配置开关功能列表

跳线名称	跳线功能	跳线连接	功能说明



跳线名称	跳线功能	跳线连接	功能说明
JP1	选择以太网模块工作接口	NF_ALE 接 CLK	NF_ALE 作为具有 SPI 接口的以太网控制芯片时钟信号
		NF_D5 接 TXD	NF_D5 作为具有 SPI 接口的以太网控制芯片数据发送引脚
		NF_D4 接 RXD	NF_D4 作为具有 SPI 接口的以太网控制芯片数据接收引脚
		SPI_CSN 连接 CSN	SPI_CSN 作为具有 SPI 接口的以太网控制芯片片选信号
		GND 接 LANCLKOUT	以太网控制芯片的 LANCLKOUT 引脚接地
		3.3V 接 3.3V	-
JP2	选择 LED 灯工作接口	IOB0 与右边接口短接	IOB0 作为 LED 灯 D11 的控制口
		IOB1 与右边接口短接	IOB0 作为 LED 灯 D12 的控制口
		IOB2 与右边接口短接	IOB0 作为 LED 灯 D13 的控制口
JP3	选择 MIC 输入或者 Line in 输入	MIC 与中间接口短接	选择 MIC 输入方式
		LINEIN 与中间接口短接	选择 Line in 输入方式
JP4	外部 Nor Flash 的片选选择	FLASH_CSN 连接 VDD33	外部 Nor Flash 的片选直接与 3.3V 电源连接
		FLASH_CSN 连接 ROMCSN	外部 Nor Flash 与 SPCE3200 芯片的 ROMCSN 连接
JP5	外部 SRAM 的片选选择	SRAM_CSN 连接 VDD33	外部 SRAM 的片选直接与 3.3V 电源连接
		SRAM_CSN 连接 ROMCSN	外部 SRAM 与 SPCE3200 芯片的 ROMCSN 连接
JP6	USB	DP	DP 接口
		DM	DM 接口
JP7	USB 主从选择	USB_DET 接 VDD5	SPCE3200 作为 USB 主机
		USB_DET 悬浮	SPCE3200 作为 USB 设备
JP8	ADC 的外部参考电压选择	ADC_ADVRT 与 AV33_ADC 连接	选择 ADC 的外部参考电压为 3.3V
Configuration Switch	JTAG、UART 及以太网配置	CFG0	闭合：不使能 ICE 断开：使能 ICE

跳线名称	跳线功能	跳线连接	功能说明
		CFG2	闭合：从内部 ROM 启动 断开：从外部 ROM（Nor Flash、Nand Flash）启动
		IOA0	闭合：IOA0 作为以太网控制芯片 INT 的控制引脚 断开：IOA0 作为通用 I/O 使用
		IOA1	闭合：IOA1 作为以太网控制芯片 WOL 的控制引脚 断开：IOA1 作为通用 I/O 使用
		UARTTX	闭合：UART 接口可发送数据 断开：UART 接口不可发送数据
		UARTRX	闭合：UART 接口可接收数据 断开：UART 接口不可接收数据
		CFG1RST ICERST	两者都闭合：CFG1RST 与 ICERST 连通 其他：CFG1RST 与 ICERST 不连通

使用说明

SPCE3200 开发板为用户提供了方便的接口，使用时，只需要用相应的连接线和外部设备连接即可。

SPCE3200 开发板通过调试器和 PC 机相连，可以完成程序的下载和在线调试，如图 5.2：用 20pin 的排线连接 SPCE3200 开发板和 SJPROBE，再通过并口线连接 SJPROBE 与 PC 机。SPCE3200 开发板的所有跳线和拨码开关按照用户自己的需求进行设置。

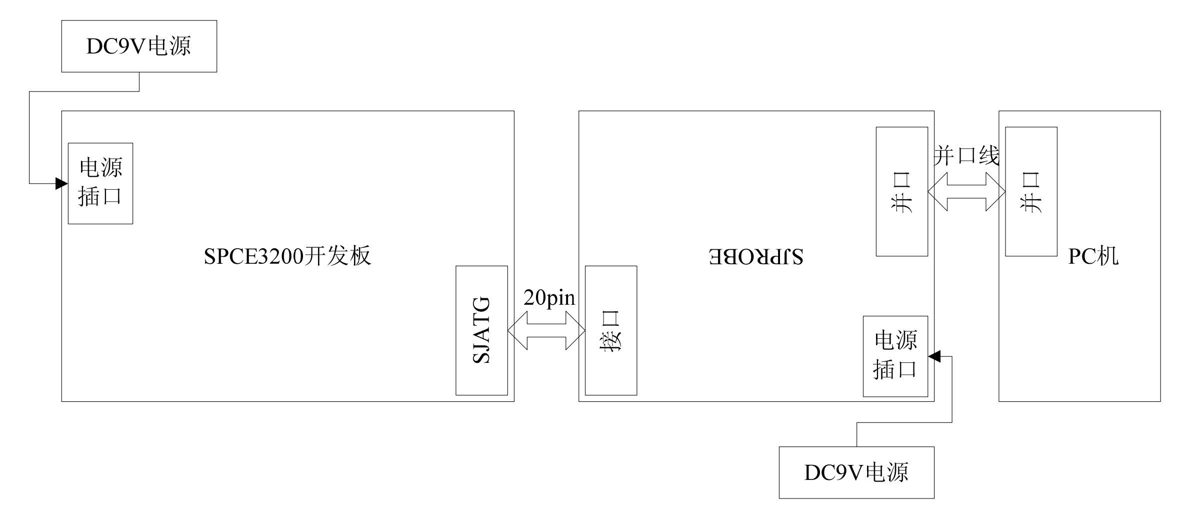


图 5.2 SPCE3200 开发板开发硬件连接



5.2 S+core IDE 集成开发环境

S+core IDE 是凌阳公司针对 SPCE3200 开发平台推出的一款功能强大的集成开发环境，支持工作区的多工程管理模式，支持 GDB 调试，为用户利用 SPCE3200 开发提供了强有力的支持；可以安装在 Windows98®、Windows2000®及 WindowsXP®上。

图 5.3 为 S+core IDE 的界面：其中菜单栏提供了用户编辑调试工程常用的功能选项；工具栏里除了常用的工具外，还提供了一些调试中常用的工具；Workspace 窗口显示工程信息，包括工程包含的源文件、资源文件、全局函数等信息；用户可以在编辑区编写程序；通过 Output 窗口可以看到工程的编译、链接、查询变量或者函数等信息。

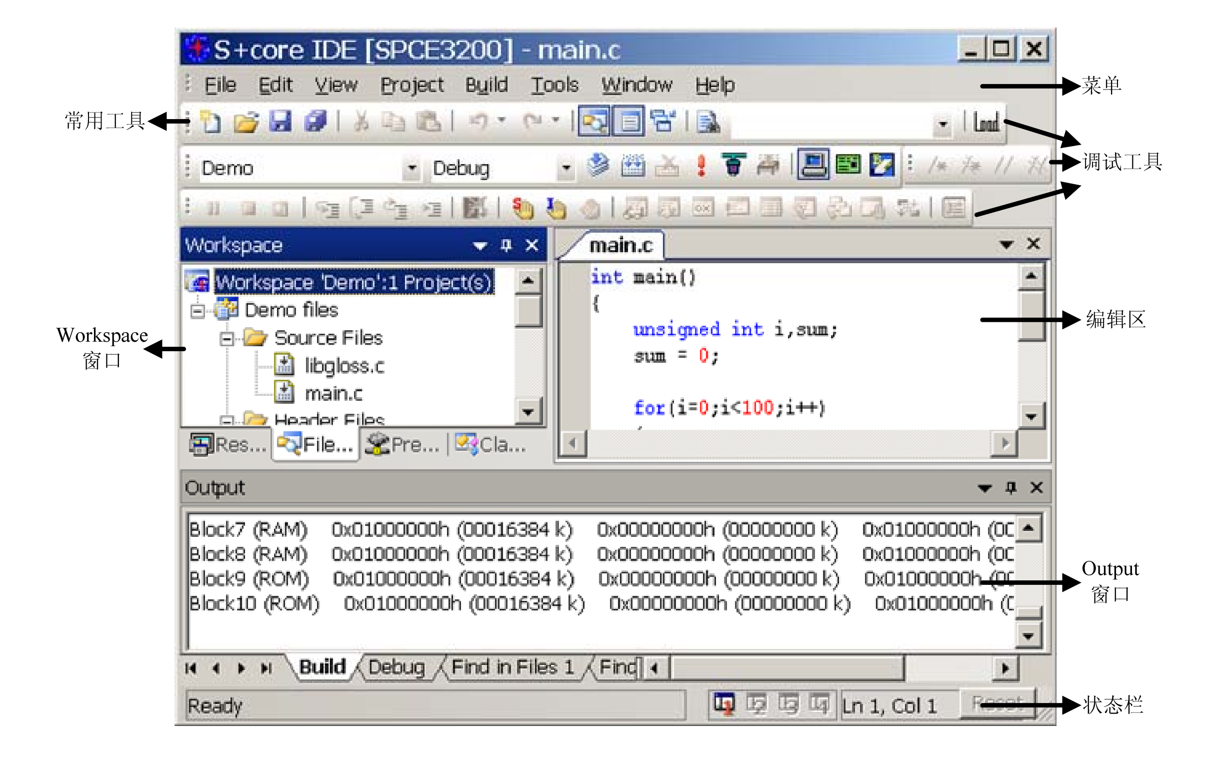


图 5.3 S+core IDE 界面

下面介绍 S+core IDE 的常用操作和常见设置。

5.2.1 工程的编辑

建立工作区

S+core IDE 在建立工程时会自动新建一个工作区，也可以在建立工程前先建立一个工作区。

如图 5.4 选中菜单的 File->New…(或者按快捷键 Ctrl+N)打开如图 5.5 的对话框。

切换到 Workspace 对话框，在“Location”栏选择工作区的保存路径；在“Workspace name”栏输入工作区名称；点击“OK”即可。

建立好的工作区界面如图 5.6。

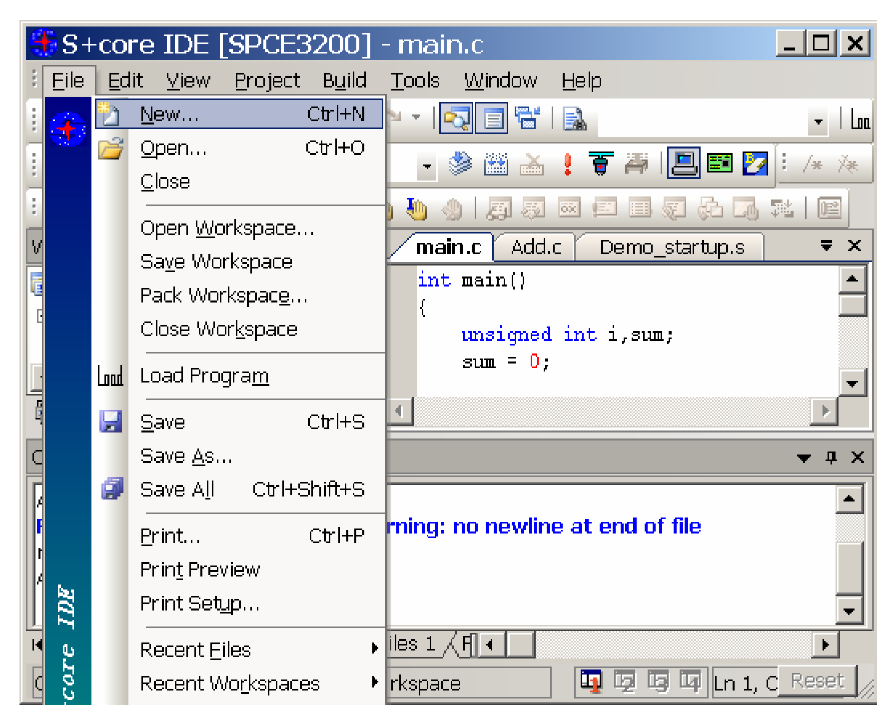


图 5.4 新建工作区/工程/文件

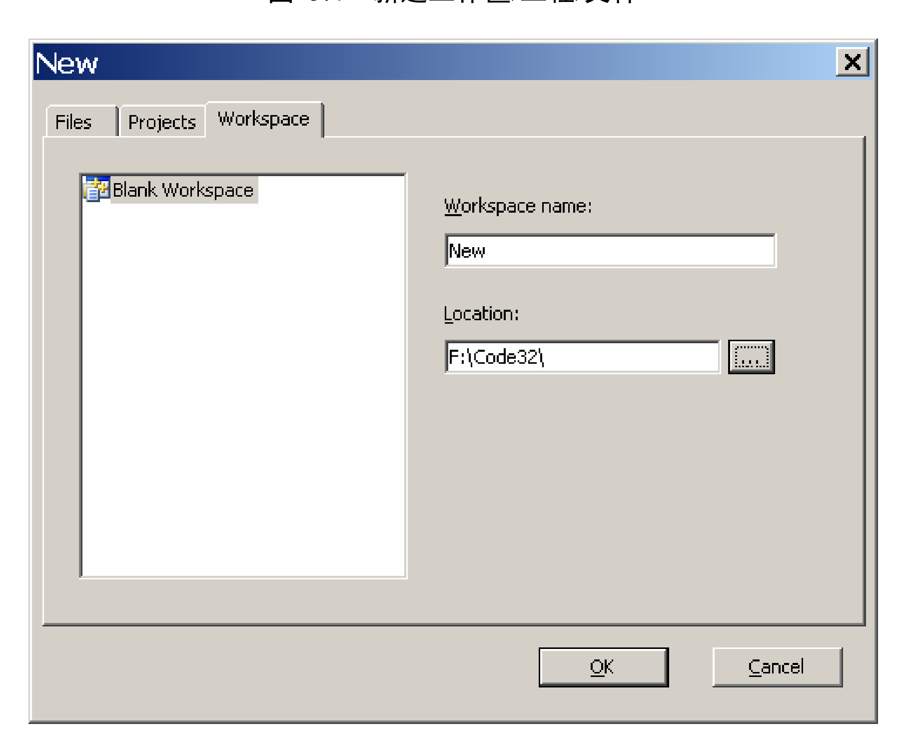


图 5.5 新建工作区

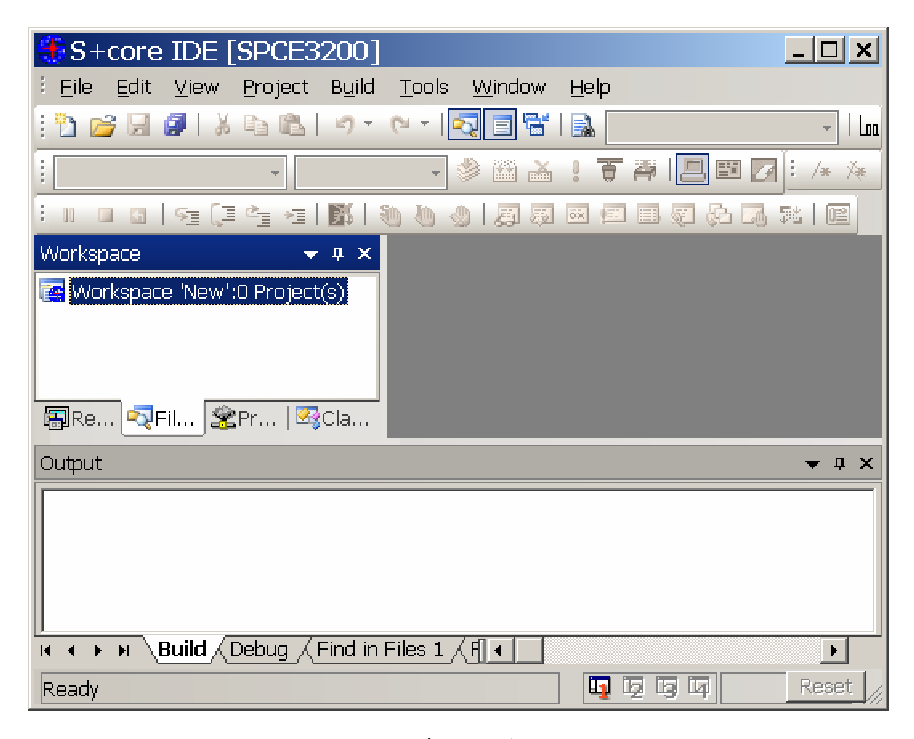


图 5.6 建立工作区界面

建立工程

如图 5.4 选中菜单的 File->New…(或者按快捷键 Ctrl+N)打开图 5.7 的对话框，在“Location”栏选择工程的保存路径；在“Project name”栏输入工程名称；如果在建立工程前已经建立工作区，选择“Add to current workspace”，否则选择“Create new workspace”。点击“OK”即可。

建立好的工程界面如图 5.8。

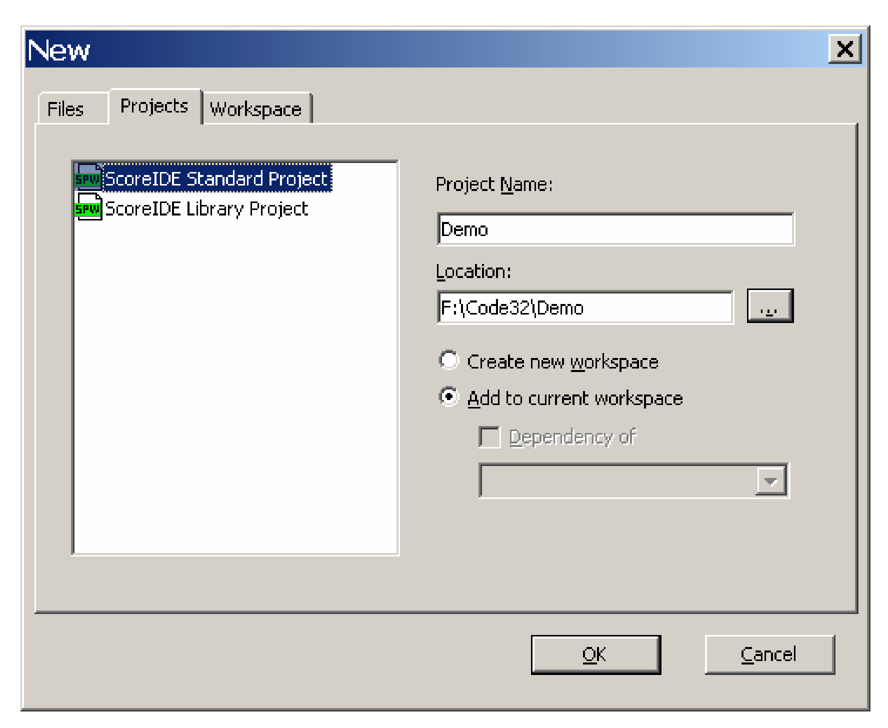


图 5.7 新建工程

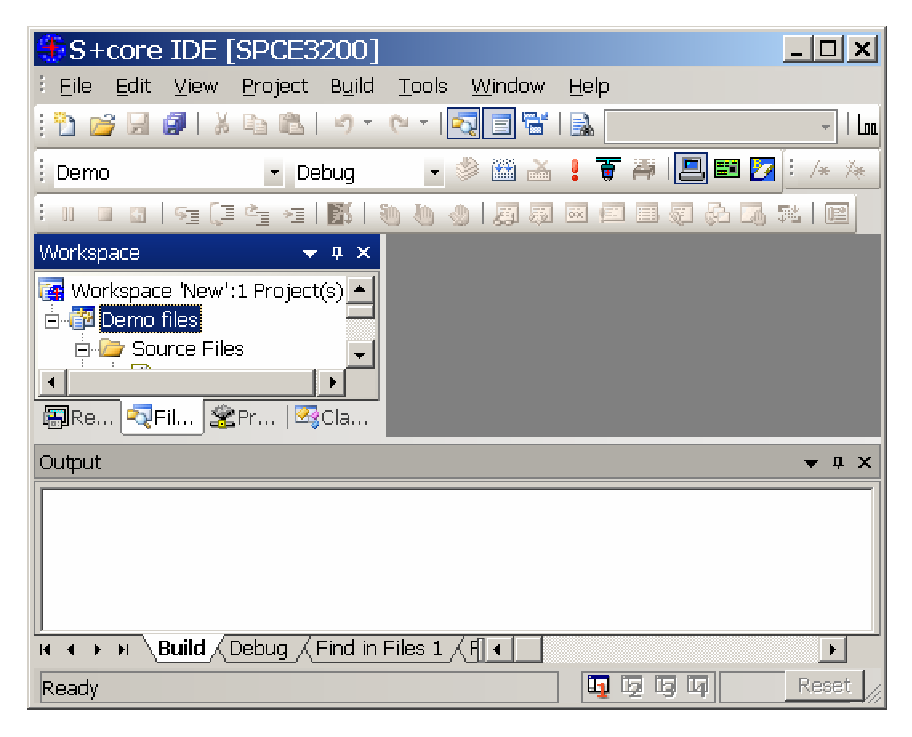


图 5.8 建立工程界面

建立文件

如图 5.4 选中菜单的 File->New…(或者按快捷键 Ctrl+N)打开图 5.9 的对话框。

选择“Add to project”在特定工程下建立文件，在“Add to project”下面的栏中可以选择同一个工作区内已经建立的工程名；

在“Location”栏选择文件的保存路径；

在“File”栏输入文件名称；

在左边的文件类型选择窗口选择要建立文件的类型（C 文件、C++文件、汇编文件或者头文件），点击“OK”即可。

建立好的文件界面如图 5.10，用户可以在编辑区建立好的文件中编写程序。

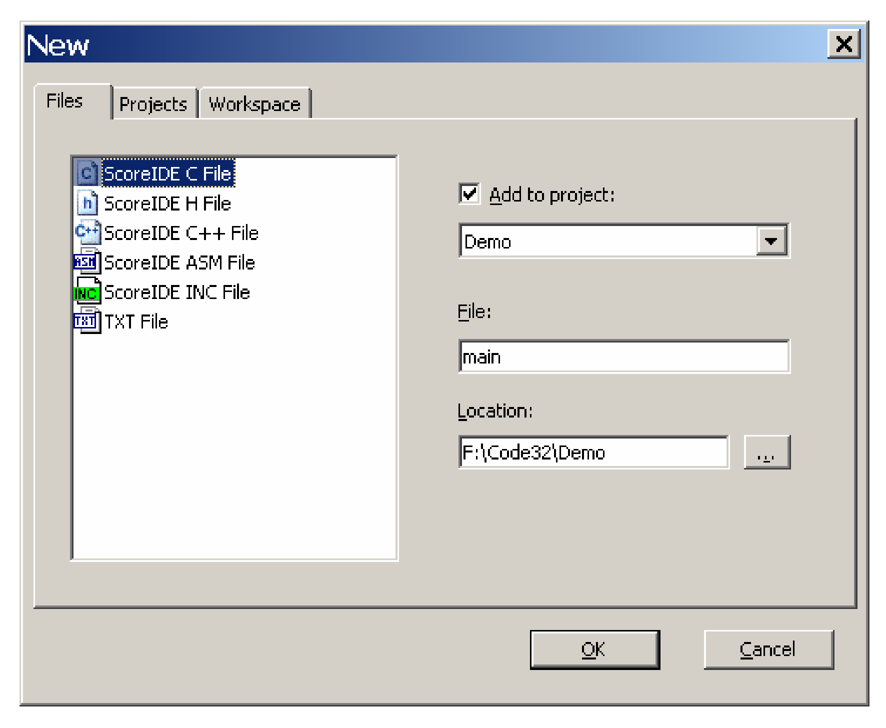


图 5.9 新建文件

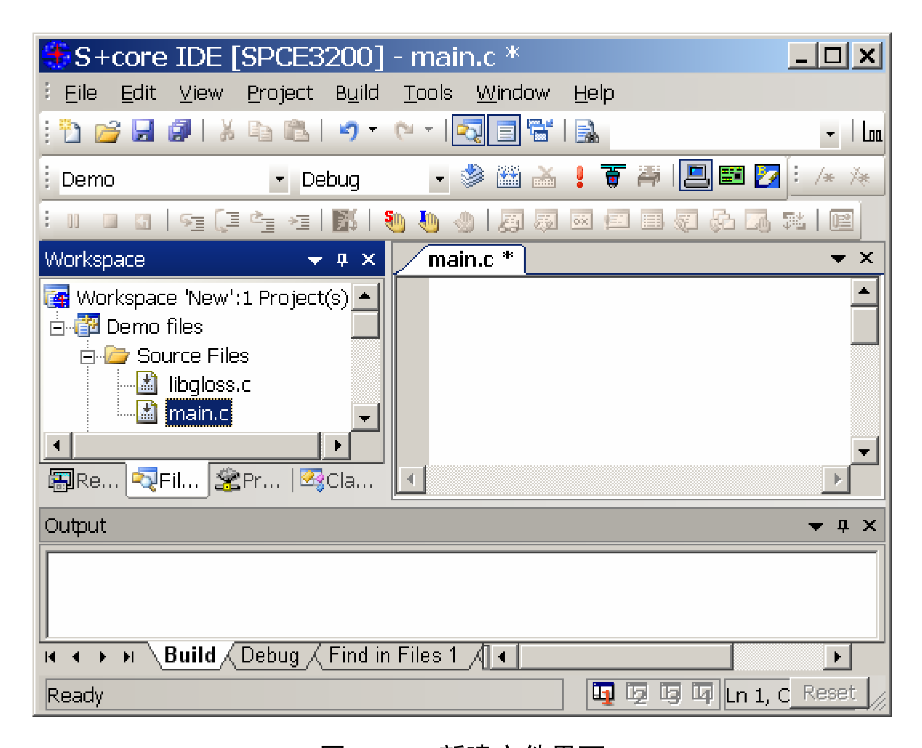


图 5.10 新建文件界面

添加工程到工作区

如图 5.11 选中菜单的 Project->Insert Project To Workspace…，打开如图 5.12 对话框。选择好要添加的工程文件后，点击打开，添加工程后的界面如图 5.13。添加后 Workspace 窗口显示的工作区工程数量比没有添加前多 1。

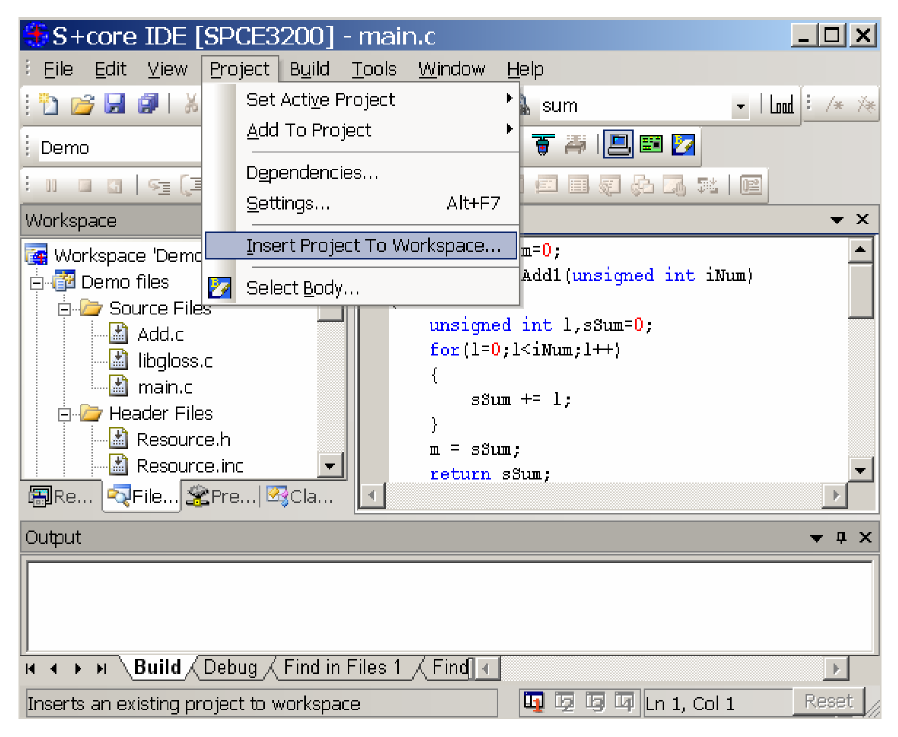


图 5.11 添加工程到工作区

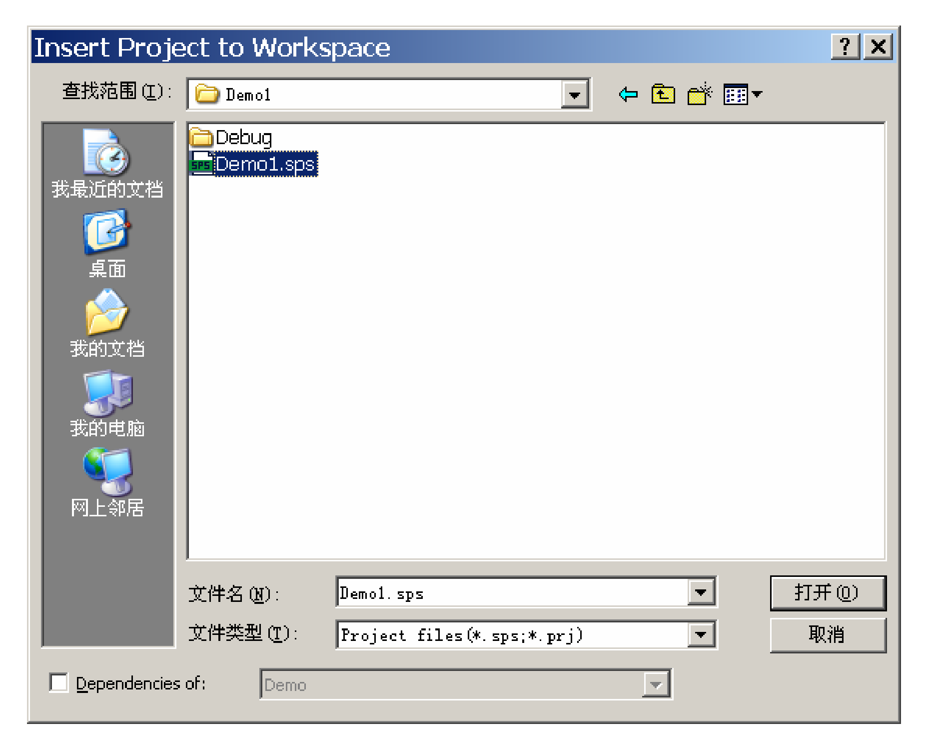


图 5.12 选择要添加的工程文件

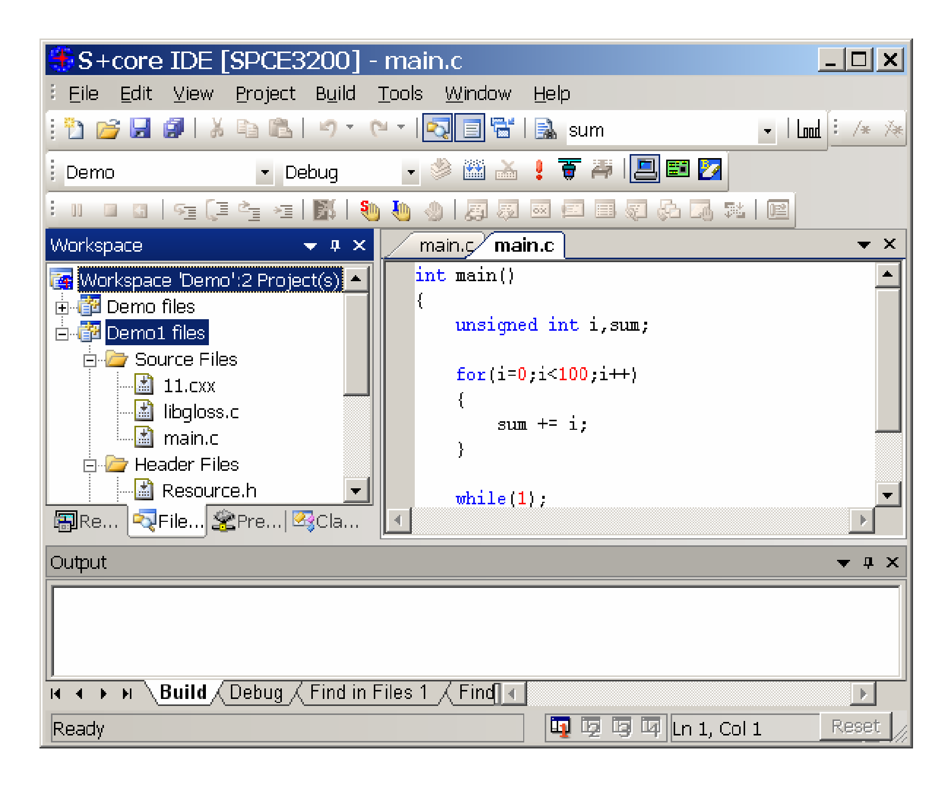


图 5.13 添加工程后界面

添加文件到工程

如图 5.14 选中菜单的 Project->Add To Project，会有两种选择“New…”和“Files…”。选择“New…”为添加一个新的文件到工程，和新建文件到工程的功能一样，不再赘述；选择“Files…”添加一个或者多个文件到工程。

如图 5.15 对话框，选择好要添加的文件后，点击打开，添加文件后的界面如图 5.16。

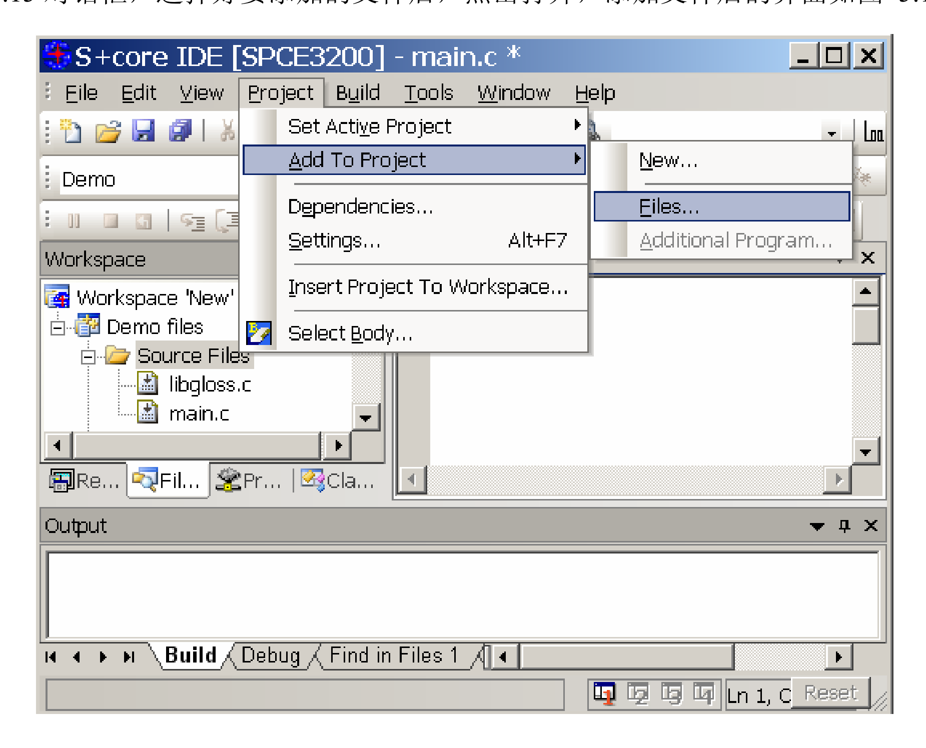


图 5.14 添加文件到工程

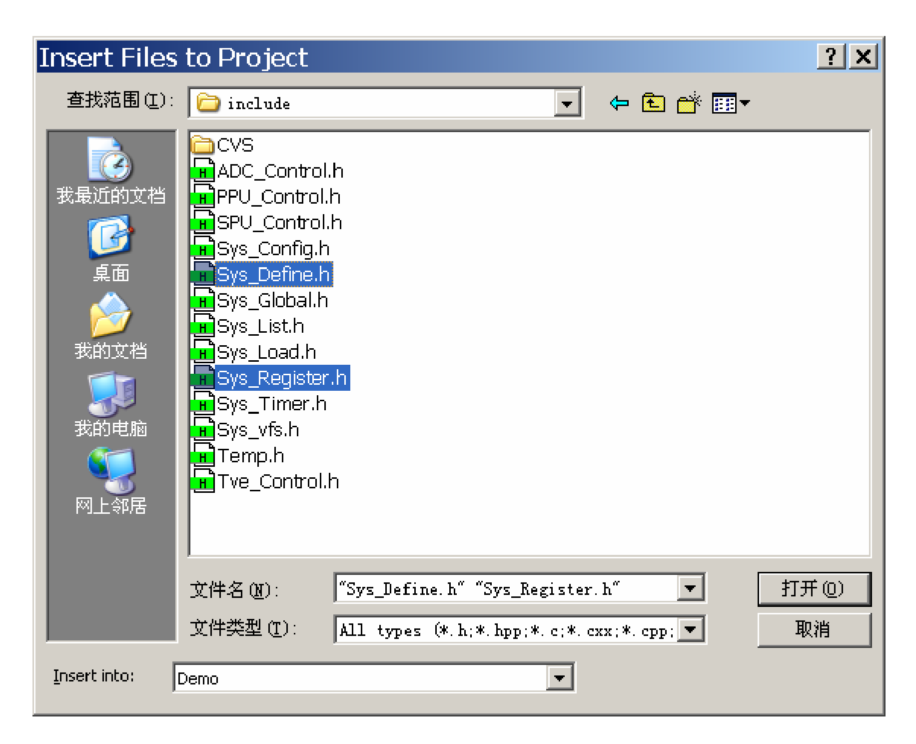


图 5.15 选择添加文件

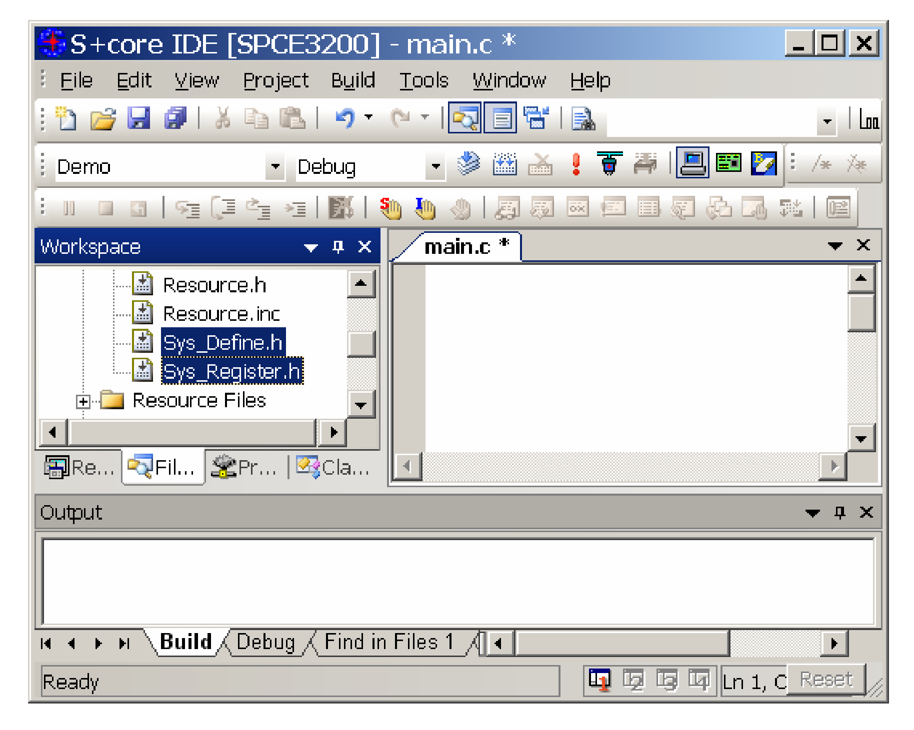


图 5.16 添加文件界面

打开已有工作区

如图 5.17，选中菜单的 File->Open Workspace…打开已有工作区；如图 5.18 选择要打开的工作区文件*.spw；打开工作区的界面如图 5.19，此时工作区包含的所有工程均被打开。

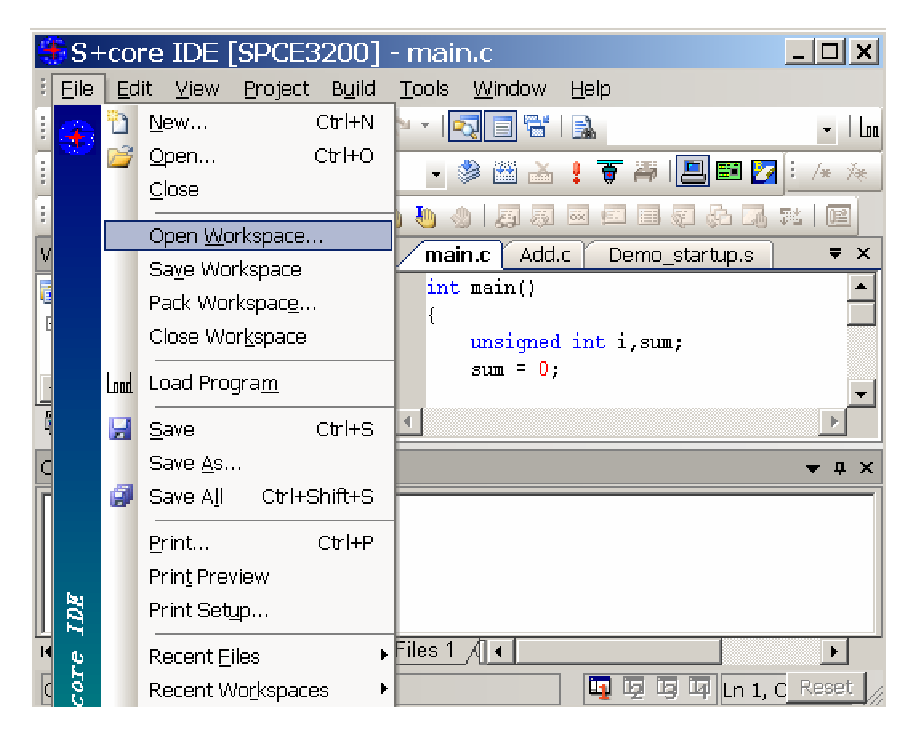


图 5.17 打开工作区



图 5.18 选择要打开的工作区

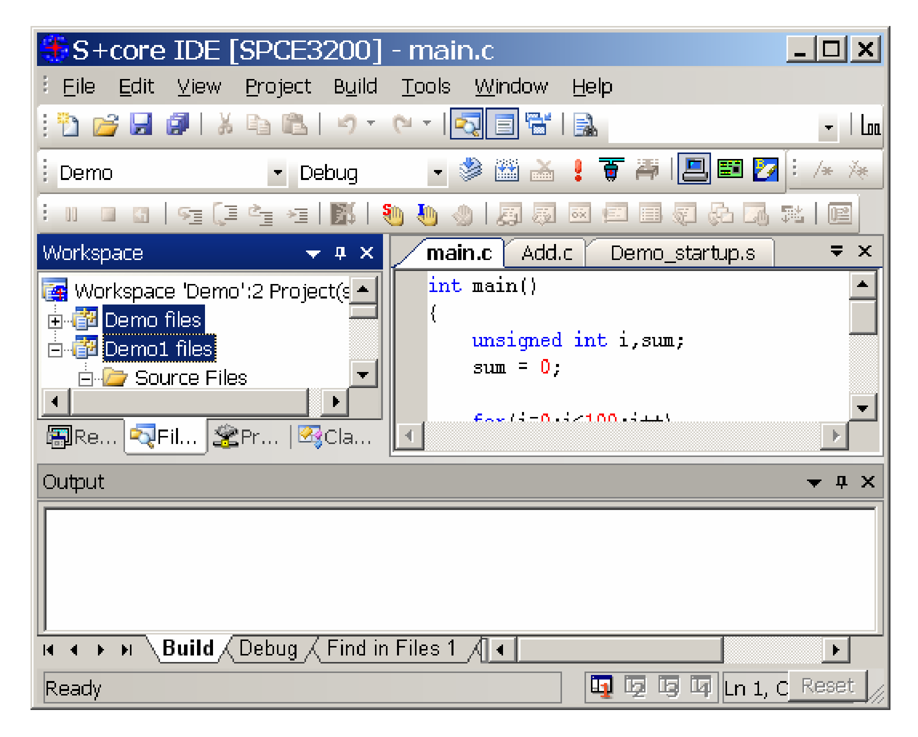


图 5.19 打开工作区的界面

打开已有工程

如图 5.20，选中菜单的 File->Open（或者按快捷键 Ctrl+O）打开已有工程或者文件；如图 5.21 选择要打开的工程文件*.sps；打开工程后的界面如图 5.22，打开时会自动生成一个和工程名相同的工作区，并在此工作区打开该工程。

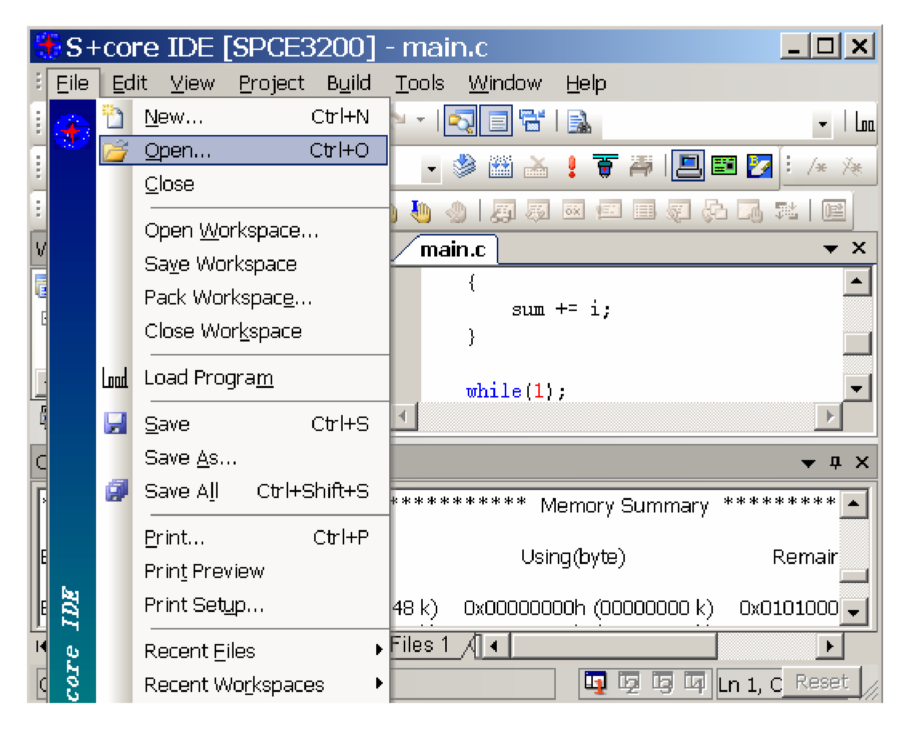


图 5.20 打开工程

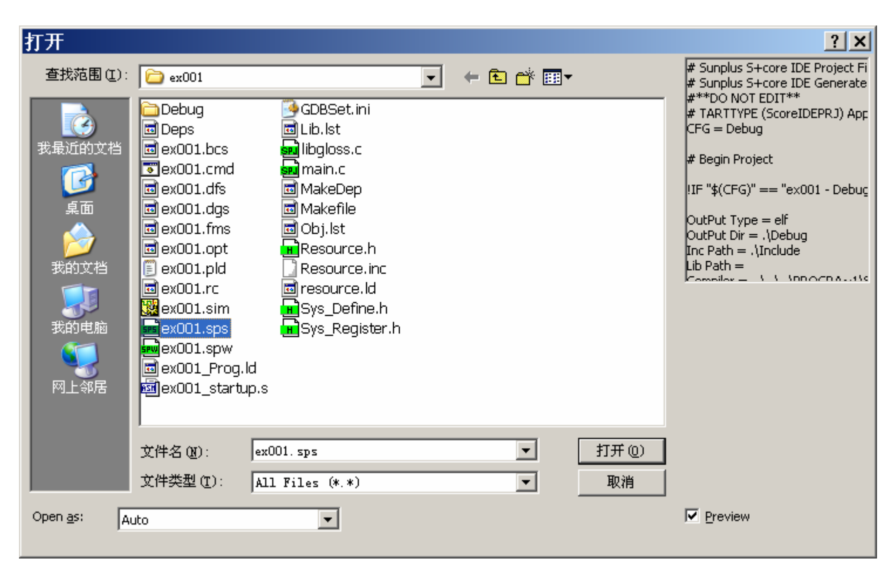


图 5.21 选择要打开的工程

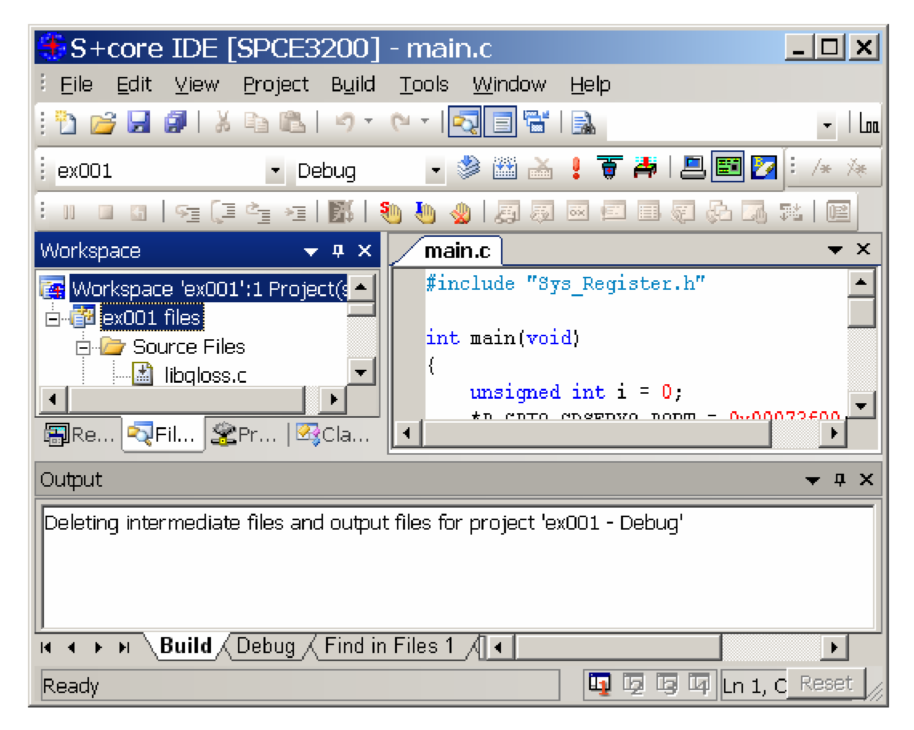


图 5.22 打开工程的界面

选择 Body

由于 S+core IDE 支持不同 S+core 内核的不同型号的芯片开发，所以在建立工程后要选择工程基于的芯片型号，即选择 Body：通过点击图标或者选择菜单 Project->Select Body…打开图 5.23 的对话框。基于 SPCE3200 开发板开发时 Body Name 要选择 SPCE3200；Probe 通常选择 Auto 即可，也可以根据自己使用的 Probe 类型选择为并口型或者 USB 型。

工程的 Body 可以改变。

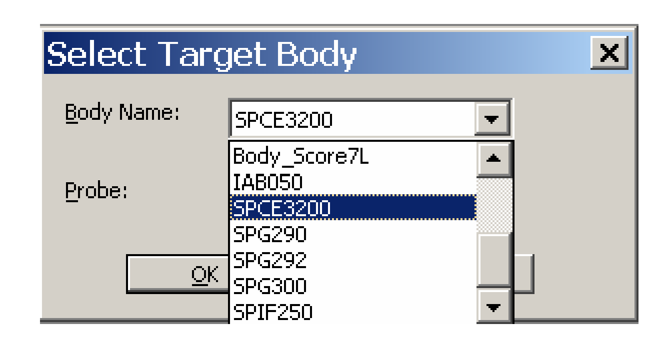


图 5.23 Body 选择

Project 的设置

如图 5.25，选择菜单 Project-> Setting…可以对工程进行设置。

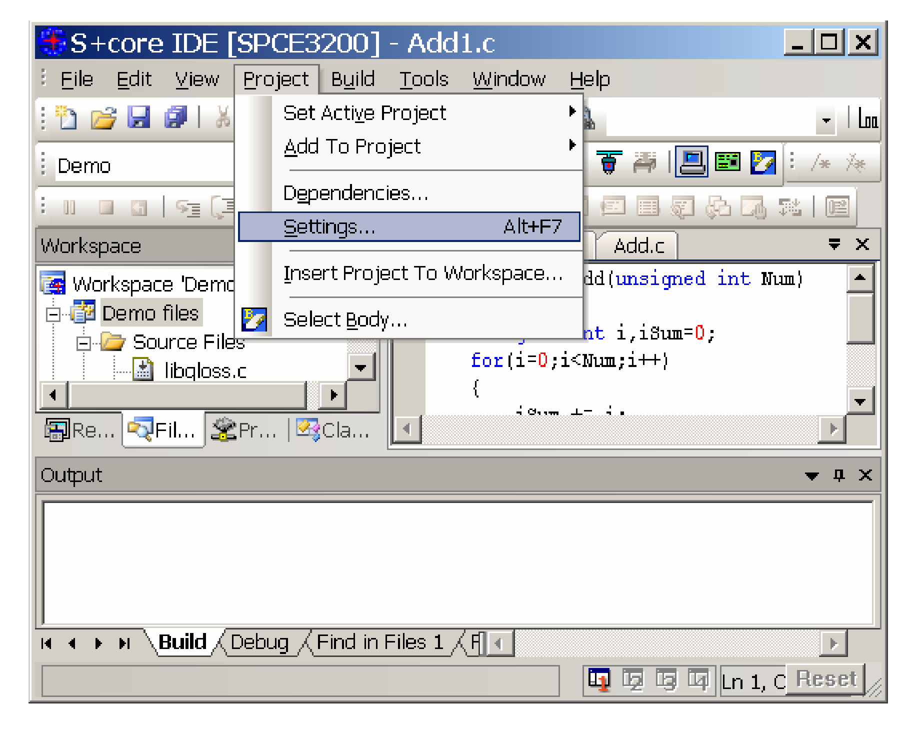


图 5.24 打开“Project Setting”

1、General

如图 5.25 对话框可以对选中工程的一般选项进行设置。图中“Category”有三种选择：选择“Common”为普通设置对话框；选择“Advanced”为高级设置对话框；选择“Exception Break”为异常中断设置对话框。

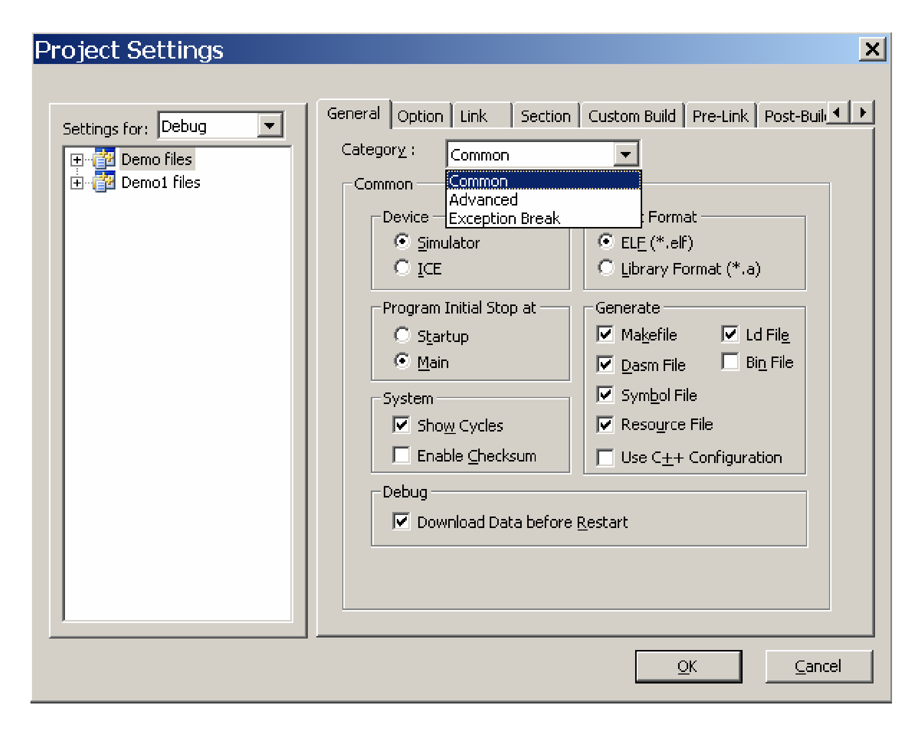


图 5.25 Project 的“General”设置

（1）Common 对话框为通用选项设置对话框，如图 5.25，其中：

“Device”选择调试方式：“Simulator”为软件仿真；“ICE”为在线调试。

“Output Format”选择输出文件的格式：*.elf 为输出可执行文件；*.a 为输出库文件。

“Program Initial Stop at”选择程序下载后停止的位置：选择 Main 停止在 main 文件中；选择 Startup 停止在*_startup.asm 文件中。

“Generate”选择是否要生成 Makefile、Symbol file、Dasm file、Resource file、Ld file 等文件；

“System”选择软件仿真时是否在状态栏里显示时钟周期，是否要检查下载数据的正确性。通常，在软件仿真时，如果选中 Show Cycles，则会在状态栏看到每一条指令的执行周期，如图 5.26。

“Debug”选择是否在重新开始运行的时候重新下载程序。

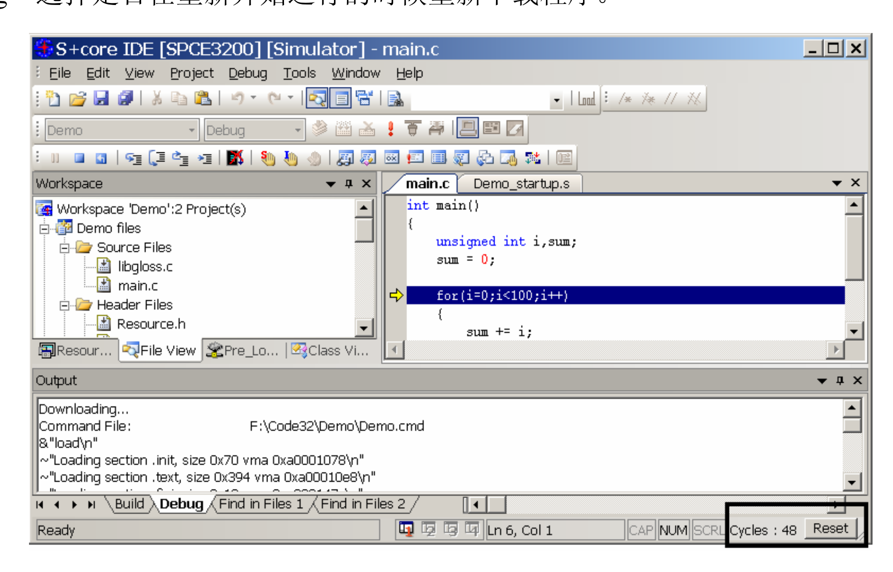


图 5.26 Show Cycles 界面

（2）Advanced 对话框为源文件映射设置等高级设置对话框，如图 5.27。

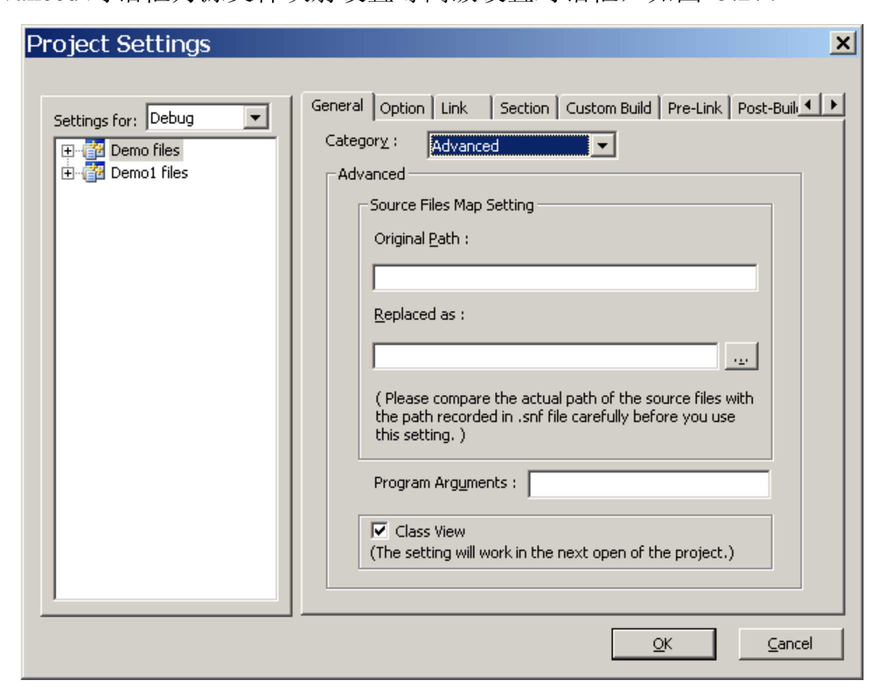


图 5.27 Advanced 对话框

其中：

“Original Path”改变路径前工作区路径。

“Replaced as”改变路径后工作区的路径。

当在本机上移动一个工程或者工作区后，通过这两栏设置好路径，不需要重新 Build 就可以下载程序。

“Class View”选择是否在 Workspace 窗口的 Class View 区显示信息；该功能只有在选择该项后，关闭重新打开工程后才起作用。

（3）Exception Break 对话框如图 5.28。该窗口设置异常的基址，默认 Exception Base Address 为 0x00000000。

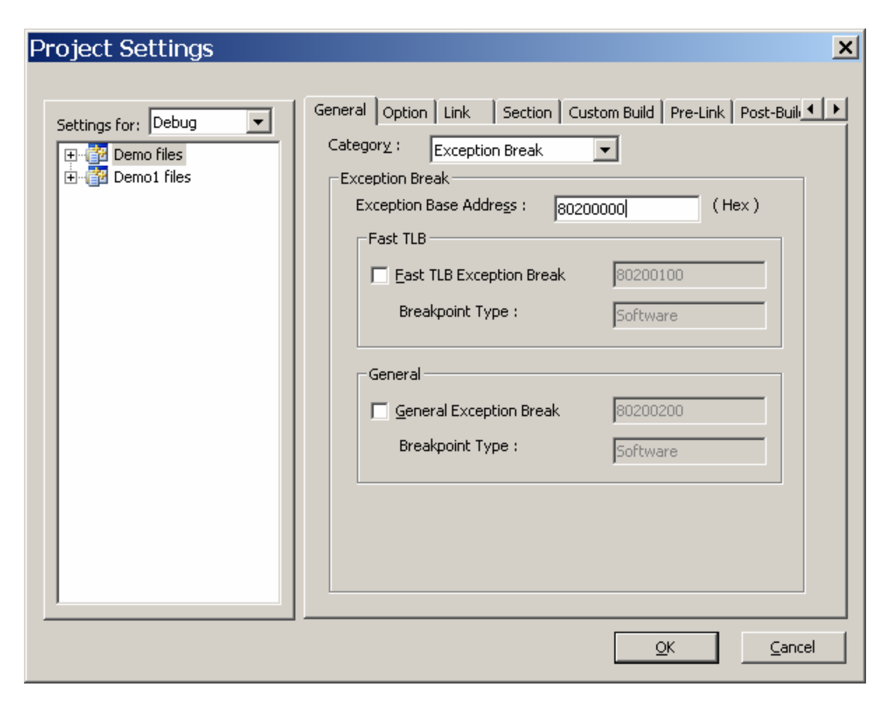


图 5.28 Exception Break 对话框

2、Option

如图 5.29 对话框，可以利用命令（有关 compiler、assembler 和 linker 的命令请参考第**错误！未找到引用源。**章）对 C compiler、assembler 和 linker 进行设置。

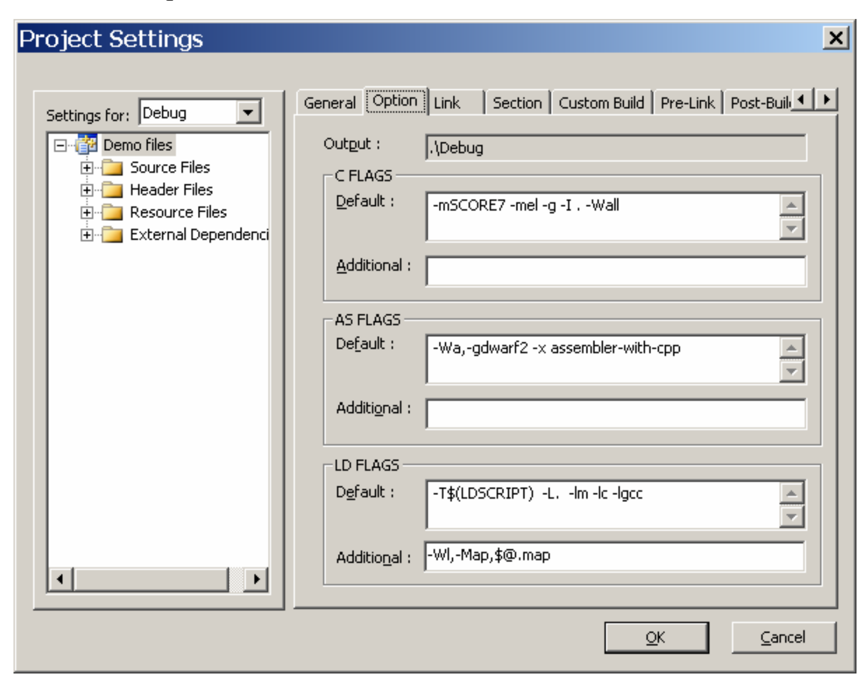


图 5.29 Project 的“Option”设置

3、Link

通过如图 5.30 对话框可以链接外部文件、库函数。

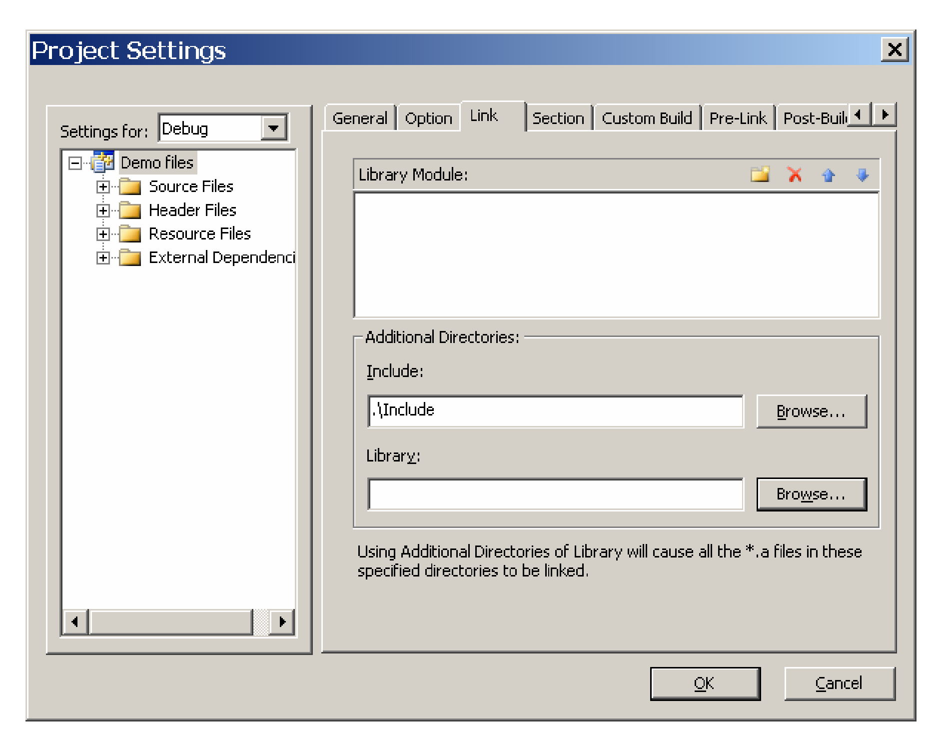


图 5.30 Project 的“Link”设置

其中各个栏的功能为：

“Library Module”：添加库函数文件。

“Include”：包含外部文件，通过“Browse…”选择要包含的文件（夹）。

“Library”：链接库函数文件，通过“Browse…”选择要链接的库文件路径，可以包含多个文件。

4、Section、Custom Build、Pre-Link、Post-Build、Simulator Setting、Simulator Memory

除了 General、Option 、Link 三个常用设置对话框外，还有六个设置对话框：Section、Custom Build、Pre-Link、Post-Build、Simulator Setting、Simulator Memory。其中：

Section 显示库/目标文件列表（Obj&Lib modules）、默认段（Default sections）、用户定义段信息（User defined sections）及其它信息（Others）；

Custom Build 为用户自定义 Build 命令对话框；

Pre-Link 为设置 Link 前编译命令对话框；

Post-Build 为设置 Build 后命令对话框；

Simulator Setting 为 Body、Cache、断点、LCD\UART\TIMER 外部接口等设置对话框，比如 Body 要选择为 Little Endian 还是 Big Endian，指令断点的最多数量设置为几个等；一般使用默认设置即可；

Simulator Memory 为 Simulator Memory 及 CPU Memory 的观察对话框。

5.2.2 工程的调试

调试工具条

S+core IDE 除了提供类似新建、打开、保存、复制、剪贴、粘贴、撤销、前进等常用工具条外，还提供了开发调试程序过程中常用到的调试工具，如图 5.31。

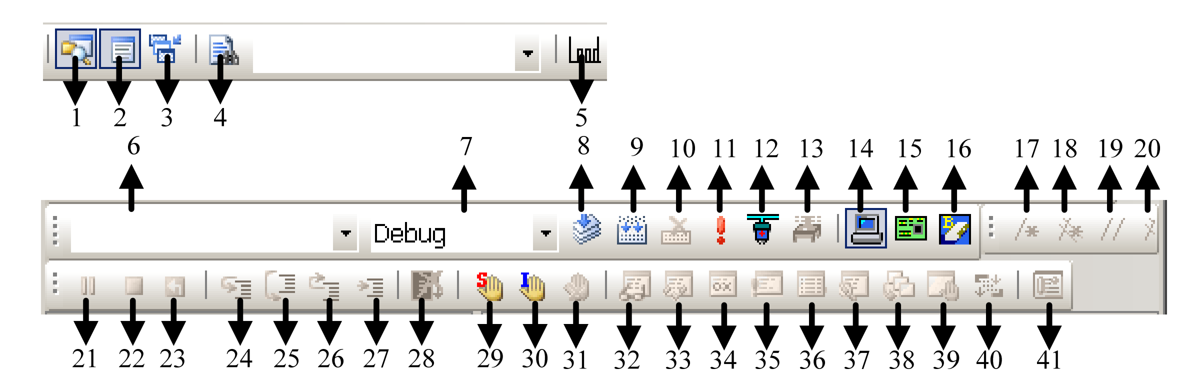


图 5.31 常用调试工具

表 5.2 为图 5.31 中各工具的功能和快捷键。

表 5.2 调试工具列表

序号	图标	快捷键	功能及使用方法
1		Alt+0	打开 Workspace 窗口；点击该图标或者按 Alt+0
2		Alt+5	打开 Output 窗口；点击该图标或者按 Alt+5
3		-	打开 Windows 窗口；点击该图标
4		Ctrl+Shift+F	查找函数、变量、字符串等；点击该图标或者按 Ctrl+Shift+F
5		-	Load 工程（可执行文件）；点击该图标
6	-	-	工程选择对话框；可以选择一个工作区的不同工程进行调试
7	-	-	选择为 Debug 模式或者 Release 模式
8		Crtl+F7	Compile 当前文件；点击该图标或者按 Crtl+F7
9		F7	Build 当前工程；点击该图标或者按 F7
10		-	停止 Build 当前工程；点击该图标
11		F5	全速运行程序；点击该图标或者按 F5
12		F8	下载程序；点击该图标或者按 F8
13		-	Debug 调试；点击该图标
14		-	选择软件仿真；点击该图标

序号	图标	快捷键	功能及使用方法
15		-	选择 ICE；点击该图标
16		-	选择 Body（*）；点击该图标
17		-	注释一段程序；选中要注释的程序，点击该图标
18		-	去掉一段程序的注释；选中被注释的程序，点击该图标
19		-	注释一行程序；选中要注释的程序，点击该图标
20		-	去掉一行程序的注释；选中被注释的程序，点击该图标
21		-	停止下载；点击该图标
22		Shift+F5	停止 Debug；点击该图标或者按 Shift+F5
23		Ctrl+Shift+F5	重新开始运行；点击该图标或者按 Ctrl+Shift+F5
24		F11	单步运行，进入子程序；点击该图标或者按 F11
25		F10	单步运行，不进入子程序；点击该图标或者按 F10
26		Shift+F11	单步运行时，跳出子程序；点击该图标或者按 Shift+F11
27		Ctrl+F10	运行到光标处；点击该图标或者按 Ctrl+F10
28		-	停止 GDB 调试；点击该图标
29		F9	在指定位置设置软件断点；点击该图标或者按 F9
30		F6	在指定位置设置指令断点；点击该图标或者按 F6
31		-	取消所有断点；点击该图标
32		Alt+6	打开 Watch 窗口；点击该图标或者按 Alt+6
33		Alt+L	打开 Local Variable 窗口；点击该图标或者按 Alt+L
34		Alt+7	打开 Registers 窗口；点击该图标或者按 Alt+7
35		Alt+8	打开 Command 窗口；点击该图标或者按 Alt+8
36		Alt+9	打开 Memory 窗口；点击该图标或者按 Alt+9
37		Alt+A	打开 Disassembly 窗口；点击该图标或者按 Alt+A
38		Alt+C	打开 Call Stack 窗口；点击该图标或者按 Alt+C
39		Alt+B	打开 Breakpoints 窗口；点击该图标或者按 Alt+B
40		Alt+R	打开 Function Browser 窗口；点击该图标或者按 Alt+R
41		Alt+O	打开 OS Information 窗口；点击该图标或者按 Alt+O



调试窗口

1、Workspace 窗口

（1）打开 Workspace 窗口

打开 Workspace 窗口有三种方法：

A、 点击图标；

B、按 Alt+0 快捷键；

C、选择菜单 View-> Workspace。

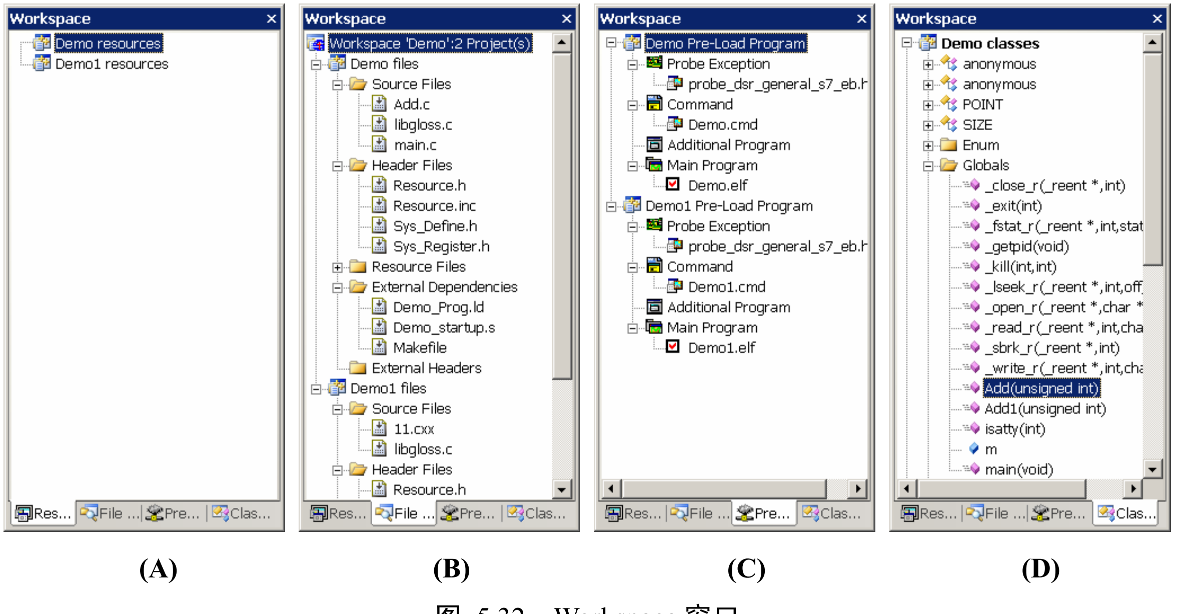
（2）Workspace 窗口介绍

A、 Resource View：如图 5.32（A），该窗口显示工作区包含工程的资源文件。

B、File View：如图 5.32（B），该窗口显示工作区包含的工程、文件信息：其中第一行显示工作区名称及该工作区包含的工程数量；从第二行开始显示工程名及工程中包含的文件名。

C、Pro_Load View：如图 5.32（C），该窗口显示预下载工程信息，包含可执行文件等。

D、 Class View：如图 5.32（D），该窗口显示工程中定义的全局变量、全局函数、宏等信息。



(A) (B) (C) (D)

图 5.32 Workspace 窗口

2、Output 窗口

（1）打开 Output 窗口

打开 Output 窗口有三种方法：

A、 点击图标；

B、按 Alt+5 快捷键；

C、选择菜单 View-> Output。

（2）Output 窗口介绍

A、 Build：如图 5.33，该窗口显示工程文件的编译或者工程的链接信息，这些信息包括程

序编译/链接过程信息、程序是否有错误、生成的可执行文件名、Memory 的占用情况等。

如果在编译或者链接时显示错误信息，双击错误信息便可以定位程序中的错误。

B、Debug：如图 5.34，该窗口显示工程下载及调试的相关信息。

C、Find in Files：如图 5.35，该窗口显示在工程中查找变量、函数、字符串等时的查找结果信息，这些信息包括变量、函数、字符串等所在的路径、文件名、在文件中的行位置、源程序语句、查找结果的总数等。

双击 Output 窗口中的查找结果便可以定位到要查找的程序语句。

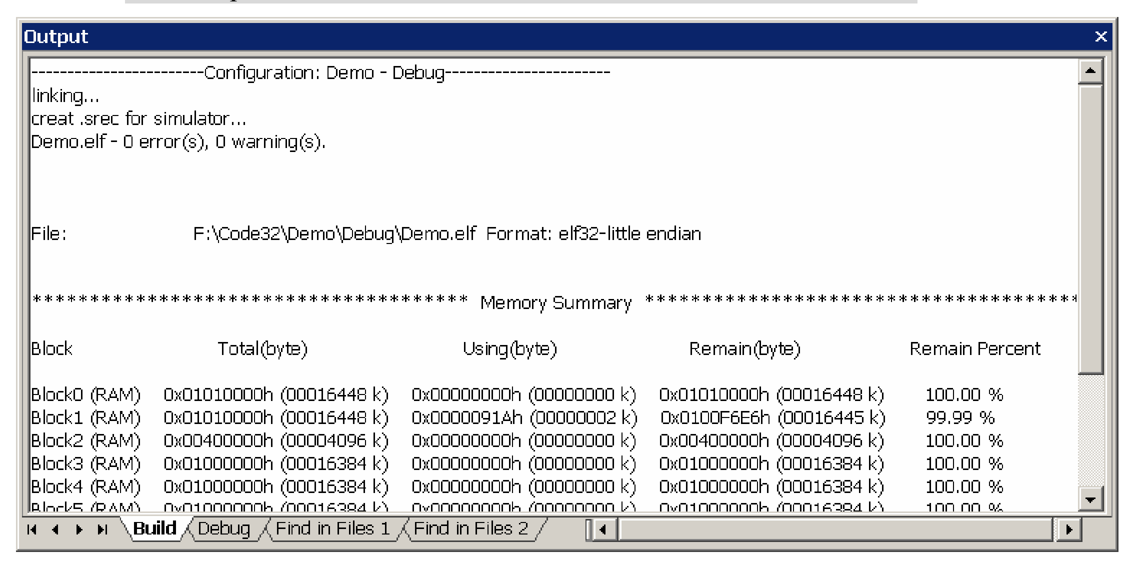


图 5.33 Output 窗口(Build)

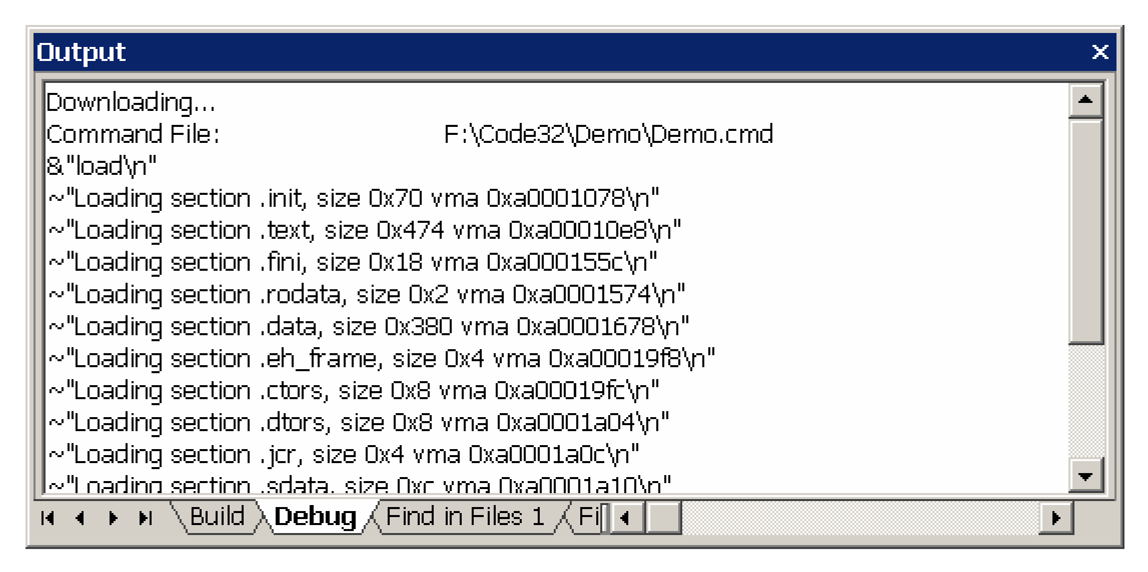


图 5.34 Output 窗口(Debug)

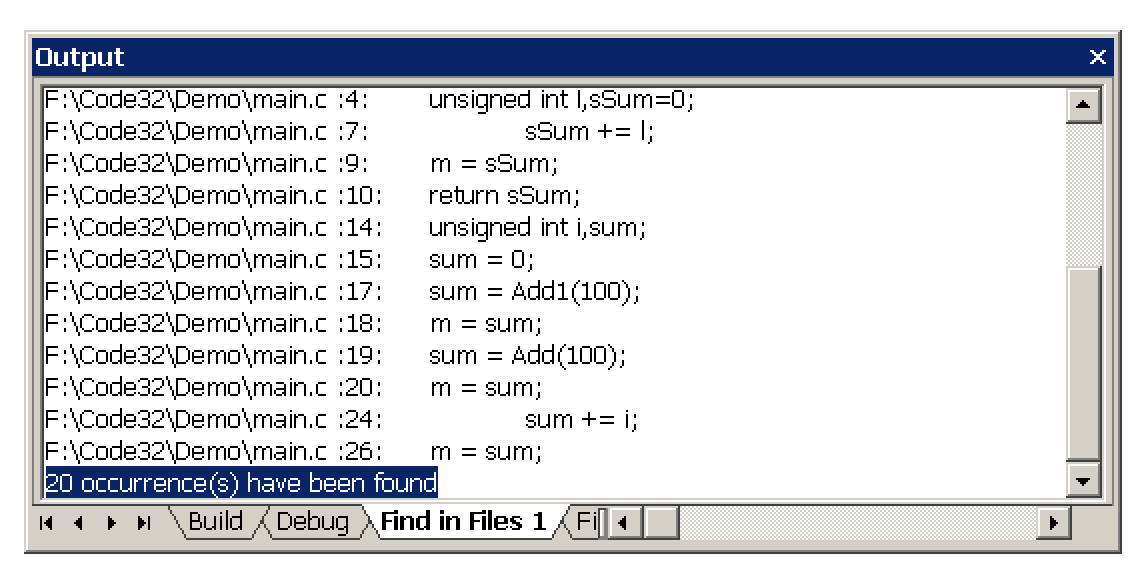


图 5.35 Output 窗口(Find in Files)

3、Windows 窗口

点击图标即可打开 Windows 窗口，如图 5.36，该窗口显示当前打开并在编辑区显示的文件，"Activate"激活文件，"Save"保存文件，"Close Window(s)"从编辑区关闭文件，"OK"退出 Windows 窗口。

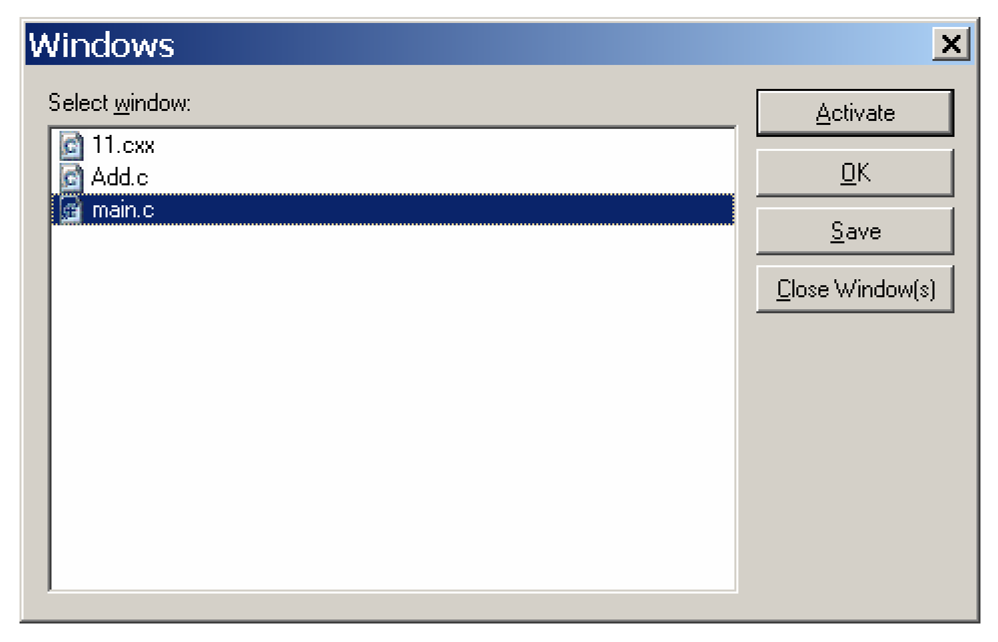


图 5.36 Windows 窗口

4、Watch 窗口

Watch 窗口为变量观察窗口。如图 5.37。

（1）打开 Watch 窗口

打开 Watch 窗口有三种方法：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+6 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Watch。

（2）Watch 窗口介绍

A、 Name：变量名称。在调试状态下，可在 Name 栏里输入变量名。

B、Value：变量值。当在 Name 栏输入变量名并按回车后，显示该变量的值。在调试的过程中，可以观察该变量值的变化；同时，还可以通过 Value 栏修改该变量的值，以便用户跟踪调试程序。

C、Address：变量地址。当在 Name 栏输入变量名并按回车后，系统会自动给该变量分配一个地址，Address 栏显示该地址值。

D、 Type：变量类型。当在 Name 栏输入变量名并按回车后，显示该变量的定义类型。

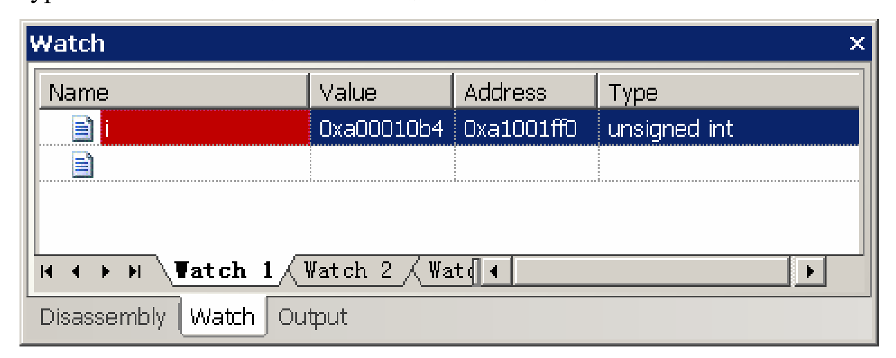


图 5.37 Watch 窗口

（3）说明

对于局部变量，只有程序运行在定义该变量的函数中时，才能显示该变量的值、地址和类型；如果程序运行到其他函数，该变量的 Value、Address 和 Type 都显示“not find”，如图 5.38。

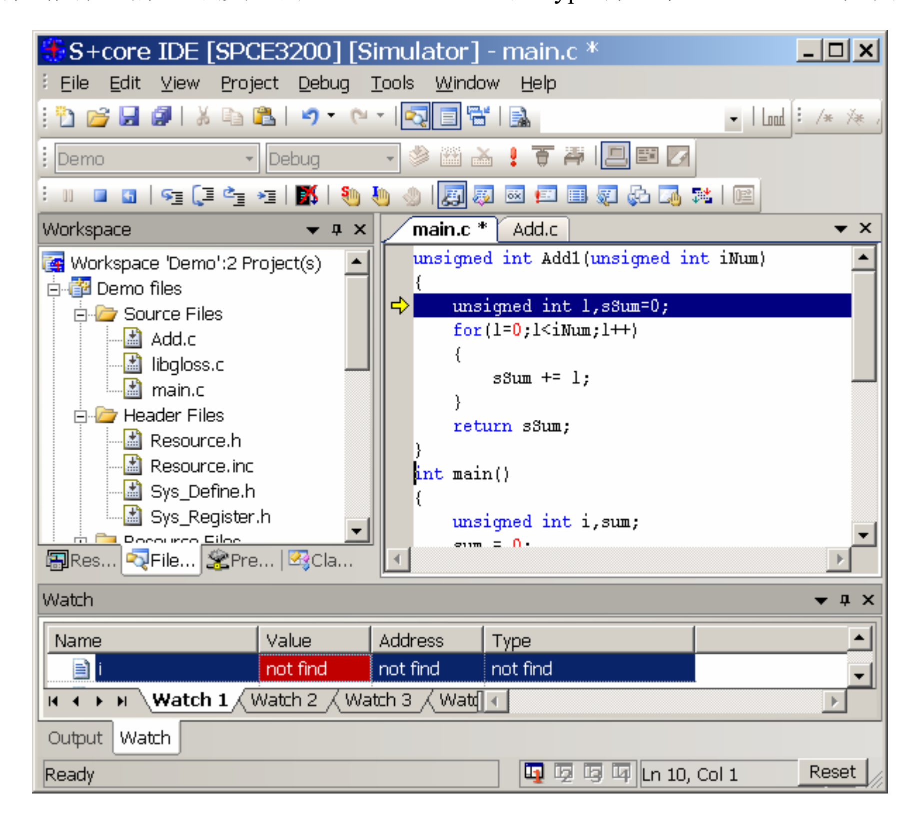


图 5.38 “i”变量观察界面

5、Local Variable 窗口

Local Variable 窗口为局部变量观察窗口。如图 5.39。

（1）打开 Local Variable 窗口

有三种方法可以打开 Local Variable 窗口：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+L 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Local Variable。

（2）Local Variable 窗口介绍

A、 Name：局部变量名称。

B、Value：局部变量值。

C、Address：局部变量地址。

D、 Type：局部变量类型。

（3）说明

Local Variable 窗口的数据不可以修改。在调试状态下，当程序运行到一个函数时，系统会自动把函数中定义的所有变量及变量的值、地址、类型显示在 Local Variable 窗口。

Local Variable 窗口不会显示全局变量，如果要跟踪全局变量，只能通过 Watch 窗口来观察。

Local Variable

Name	Value	Address	Type
i	0xa00010b4	0xa1001ff0	unsigned int
sum	0x00000000	0xa1001ff4	unsigned int

图 5.39 Local Variable 窗口

6、Registers 窗口

Registers 窗口为寄存器观察窗口。Registers 窗口包含两个窗口：General 窗口和 TLB Entry 窗口，如图 5.40 和图 5.41。

（1）打开 Registers 窗口

有三种方法可以打开 Registers 窗口：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+7 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Registers。

（2）说明

Registers 窗口显示各模式下寄存器的数据，在程序运行过程中，以红色字表示该寄存器的数据被修改。在调试过程中，这些寄存器的数据也可以修改。

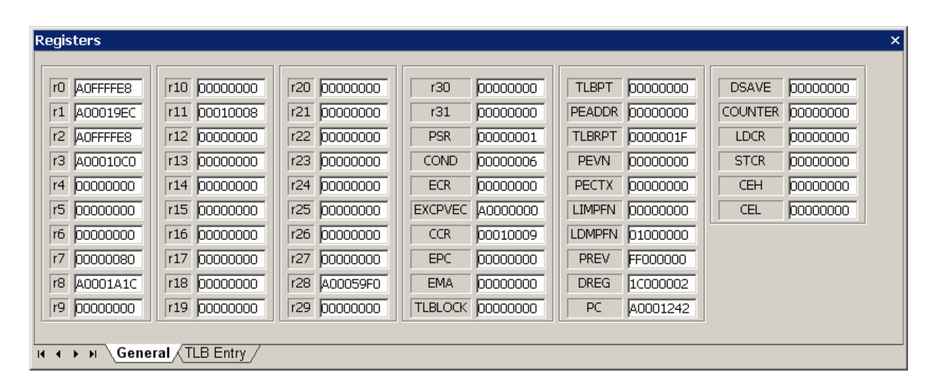


图 5.40 Registers(寄存器)观察窗口

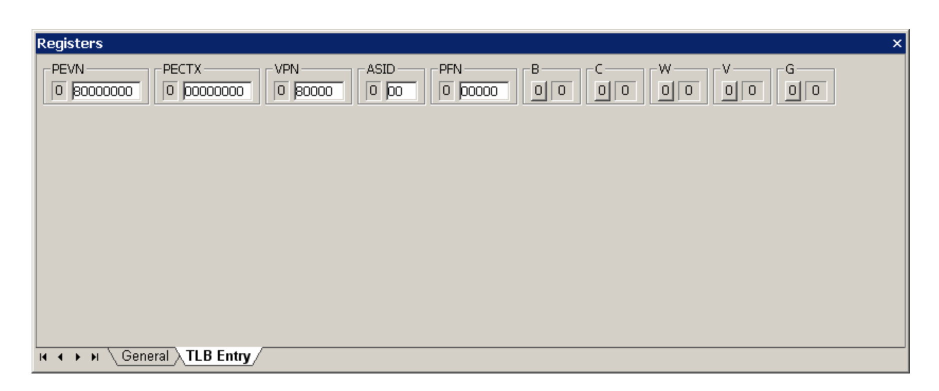


图 5.41 Registers(寄存器)观察窗口

7、Command 窗口

Command 窗口为 S+core IDE 命令窗口。如图 5.42。

（1）打开 Command 窗口

有三种方法可以打开 Command 窗口：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+8 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Command。

（2）说明

调试过程中，通过这个窗口可以执行一些 GDB 命令，例如图 5.42 中提示的“cls”，当在“>”后加 cls 后回车，就可以看到 Command 窗口内容被清除，重新显示刚打开时的信息。

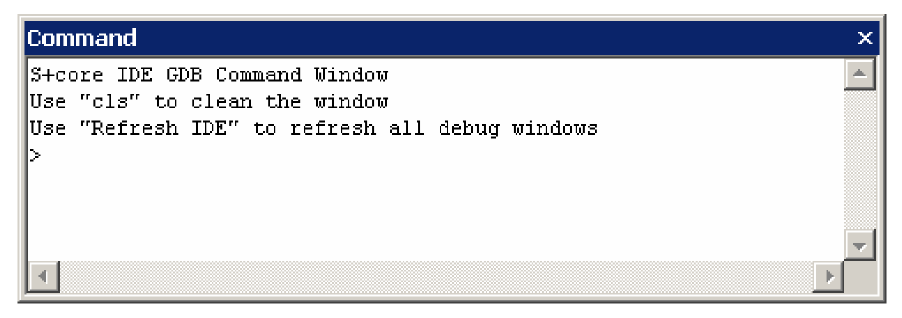


图 5.42 Command 窗口

8、Memory 窗口



Memory 窗口存储器观察窗口。如图 5.43。

（1）打开 Memory 窗口

有三种方法可以打开 Memory 窗口：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+9 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Memory。

（2）Memory 窗口介绍

A、 Memory 下拉菜单：该菜单可以选择 RAM 或者 ROM 的地址段。系统按照存储器的映射情况把存储器分为不连续的 16 个不同大小的 Block，其中第 0~8Block 为 RAM 地址，第 9、10Block 为 ROM 地址，第 11~15Block 为保留区。在 Block 选择框后面的框分别显示选择 Block 是属于 RAM 还是 ROM，同时显示该 Block 的起始和结束地址。当通过 Memory 下拉菜单选择好 Block 后，地址内容区就会显示从该 Block 起始地址开始的存储数据。

B、Address 地址输入：通过该栏可以输入一个地址，按回车后，地址内容区显示从该地址开始的数据，同时 Memory 下拉菜单自动显示当前地址所在的 Block，该 Block 所在的存储区（RAM 或 ROM）、起始地址和结束地址。

Address 地址输入栏只能输入 Block0~10 中的任意一个地址，否则系统会提示输入了一个无效的地址。

C、地址内容：地址内容区最左边一列显示地址；中间显示这些地址单元的数据的 16 进制数据；最右边显示这些地址单元数据的 ASCII 码。

（3）说明

在调试过程中，Memory 窗口地址内容区的数据可以修改。

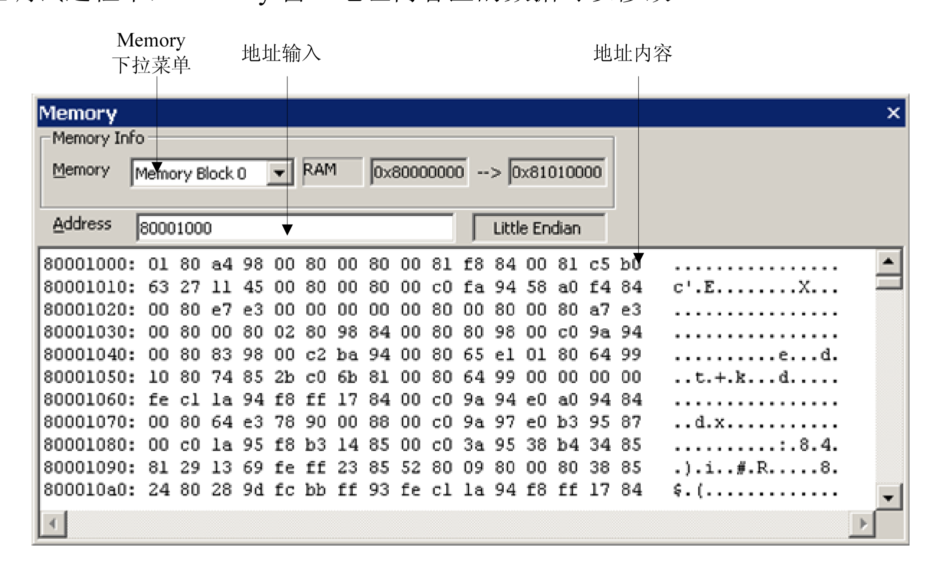


图 5.43 Memory 窗口

9、Disassembly 窗口

Disassembly 窗口为反汇编观察窗口。如图 5.44。

（1）打开 Disassembly 窗口

有三种方法可以打开 Disassembly 窗口：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+A 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Disassembly。

（2）说明

调试过程中，通过 Disassembly 窗口可以看到程序的反汇编代码，如图 5.44，黑色字体的部分为源代码所在的路径、程序在文件中的行位置及源程序；灰色的部分为反汇编代码。

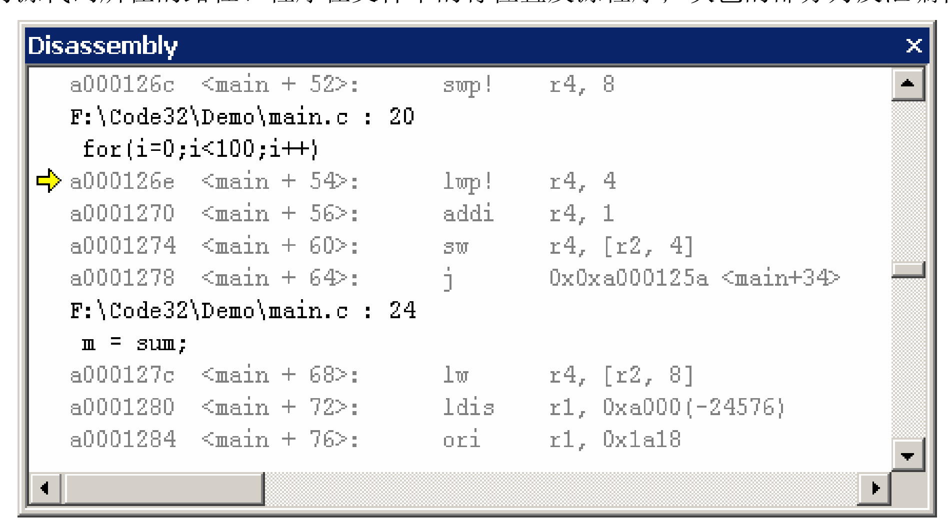


图 5.44 Disassembly 窗口

10、Call Stack 窗口

Call Stack 窗口为堆栈调用情况观察窗口。如图 5.45。

（1）打开 Call Stack 窗口

有三种方法可以打开 Call Stack 窗口：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+C 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Call Stack。

（2）说明

如图 5.45，在调试过程中，通过 Call Stack 窗口可以看到当前运行程序的堆栈调用情况，调用函数时，先把该函数压入堆栈，等到函数返回时，再把该函数弹出堆栈。

窗口显示的内容中左边的部分为函数名及其参数，右边的部分为程序运行的行位置。

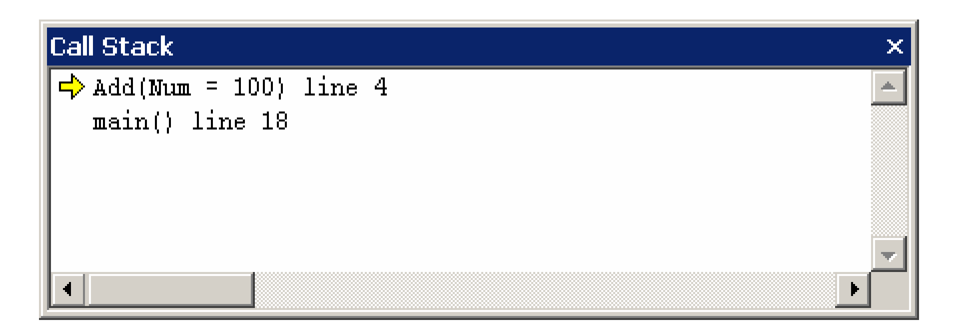


图 5.45 Call Stack 窗口

11、Breakpoints 窗口

Breakpoints 窗口为断点设置及观察窗口，包含三个窗口：软件断点、指令断点和指定变量断点（数据断点）窗口，如图 5.46~图 5.48。

（1）打开 Breakpoints 窗口

打开 Breakpoints 窗口有三种方法：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+B 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Breakpoints。

（2）Breakpoints 窗口介绍

A、 Address：断点设置地址。

B、Condition：断点条件。

C、Enable：使能断点。

D、 Line：断点所在程序文件中的行位置。

E、File Name：断点所在文件名称及路径。

F、 Variable：变量断点。

G、 “ADD”：增加断点。

H、 “Modify”：修改断点。

I、 “Remove”：移除选中断点。

J、 “Remove All”：移除所有断点。

（3）说明

如图 5.46，在通过 Breakpoints 窗口设置软件断点时，先在 Address（Hex）栏里填写需要设置断点的程序地址（如果有条件，再在 Condition 栏里填写条件），点击“Add”，在右边的框内会显示该断点的信息，在 Enable 栏选中，就可以完成软件断点设置。同时，通过其他方式设置的断点也在 Breakpoints 窗口显示，也可以通过该窗口进行使能、不使能、修改、移除等操作。

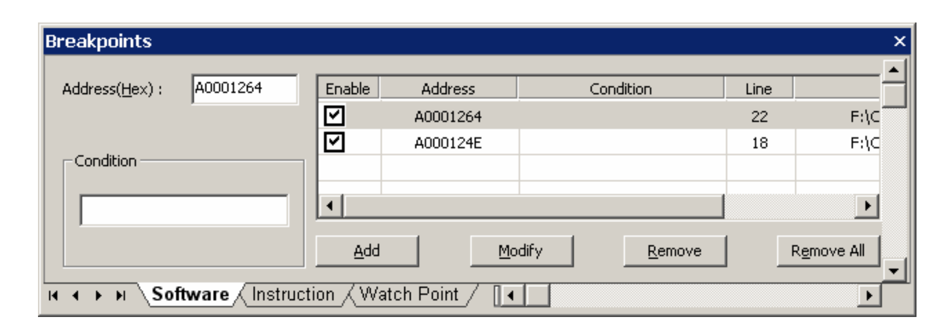


图 5.46 Breakpoints 窗口(Software)

如图 5.47，通过 Breakpoints 窗口设置指令断点的方法和设置软件断点的方法相同。另外，通过其他方式设置的断点也在 Breakpoints 窗口显示，也可以通过该窗口进行使能、不使能、修改、移除等操作。

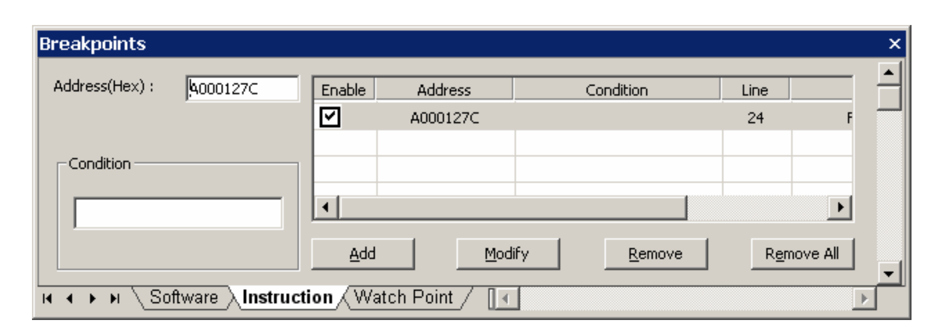


图 5.47 Breakpoints 窗口(Instruction)

如图 5.48，通过 Breakpoints 窗口设置变量断点的方法和上面两种类似：先在 Variable 栏里填写变量名，点击“Add”，在右边的框内会显示该断点的信息，在 Enable 栏选中，就可以完成变量断点设置；此时全速运行时系统会在有该变量的地方停下。如果在设置断点的同时在 Condition 栏加上条件（= =、!=、>、>=、<、<=），比如变量等于（= =）一个数据，此时全速运行时系统会在有该变量等于设置数据的时候停下，这对于一些循环来说，是非常有用的。

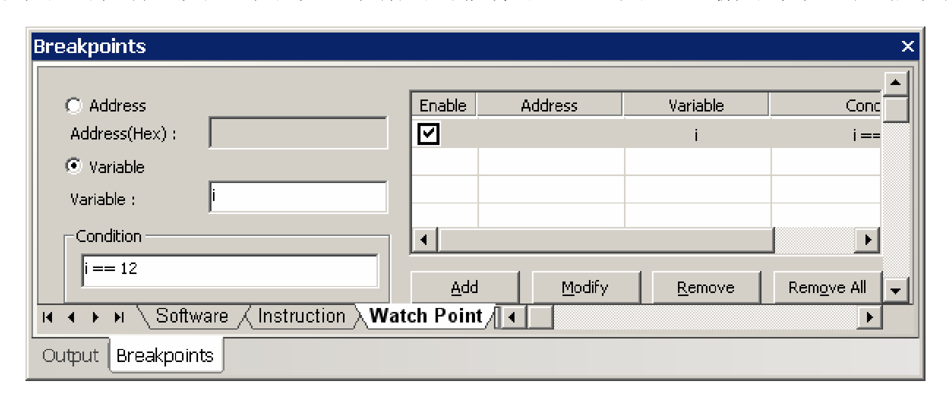


图 5.48 Breakpoints 窗口(Watch Point)

12、Function Browser 窗口

Function Browser 窗口为功能函数观察窗口，如图 5.49。

（1）打开 Function Browser 窗口

打开 Function Browser 窗口有三种方法：

A、 在调试状态下，点击图标；

B、在调试状态下，按 Alt+R 快捷键；

C、在调试状态下，选择菜单 View->Debug Windows->Function Browser。

（2）Breakpoints 窗口介绍

A、 Files：当前工程所有的 C 语言文件列表，通过 Select 栏选择或者不选择 C 文件，如果要选择，在方框上点击选中；Index 为序号；Files 栏显示 C 语言文件名及路径。

B、Function：选中的 C 文件定义的所有功能函数列表，通过“Fast Input”栏输入查找指定 C 文件中的功能函数；Index 为序号；Name 栏显示功能函数名及其参数；File Name 栏显示功能函数所在的文件名及路径。

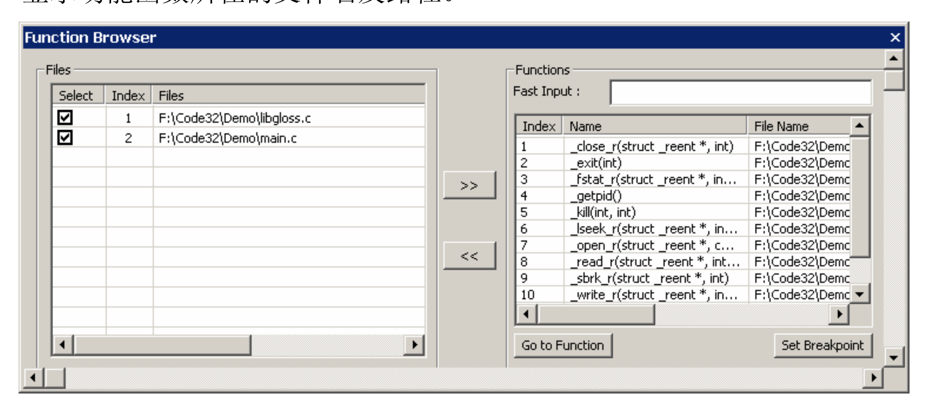


图 5.49 Function Browser 窗口

13、OS Information 窗口

OS 信息窗口，在 OS 的相关章节会有详细介绍，故不做阐述。

5.3 应用举例

例 5.1：在 S+core IDE 下建立一个工作区 MyCode，并在其中建立两个工程 Code1 和 Code2，Code1 实现的功能是从 1 加到 100，计算出累加结果；Code2 实现的功能是使 SPCE3200 开发板的 LED1 灯闪烁。

要求：

软件仿真调试工程 Code1

1、单步运行，观察执行各条程序语句需要的周期数，并分别通过变量观察窗口观察各定义变量的变化、寄存器观察窗口观察各寄存器的变化；

2、全速运行，分别添加一个软件断点、一个指令断点和一个变量断点；通过添加的变量断点观察从 0 加到 50 时的结果。

3、全速运行，通过变量观察窗口观察最后的运行结果。

利用 ICE 进行在线调试工程 Code2。

1、单步运行，通过 Memory 窗口观察 0x8820004C~0x8820004F 地址单元的内容的变化和寄存器窗口观察寄存器的变化；

2、全速运行，观察其现象。

解析及步骤：

第一步：新建工作区 MyCode，如图 5.50。

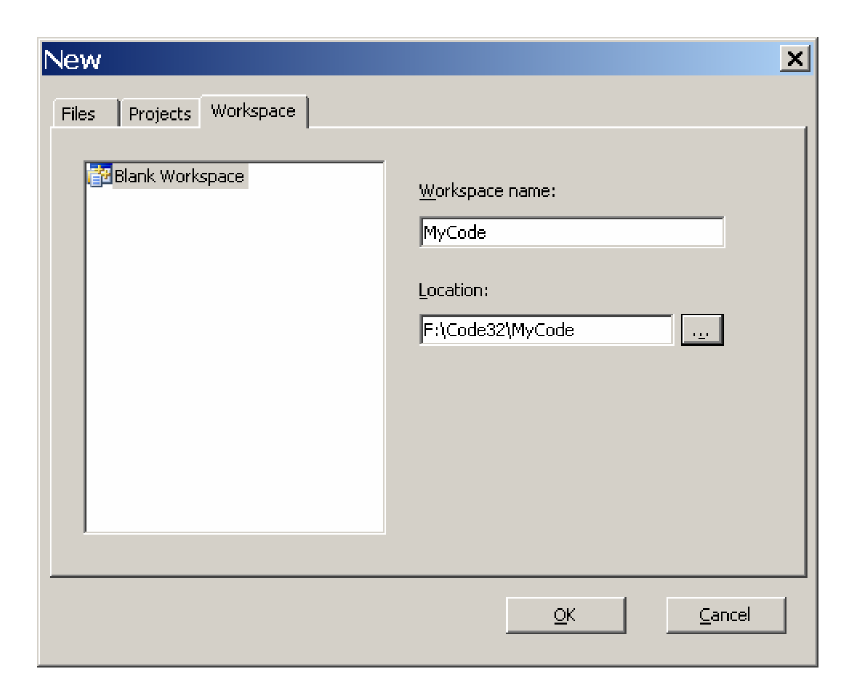


图 5.50 新建工作区 MyCode

第二步：新建工程 Code1 和 Code2，如图 5.51 和图 5.52。

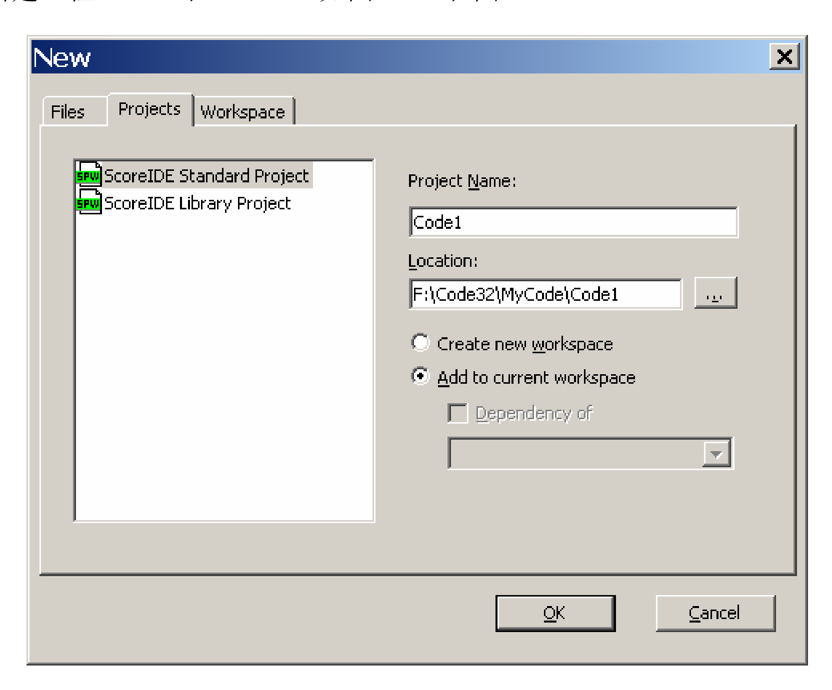


图 5.51 新建工程 Code1

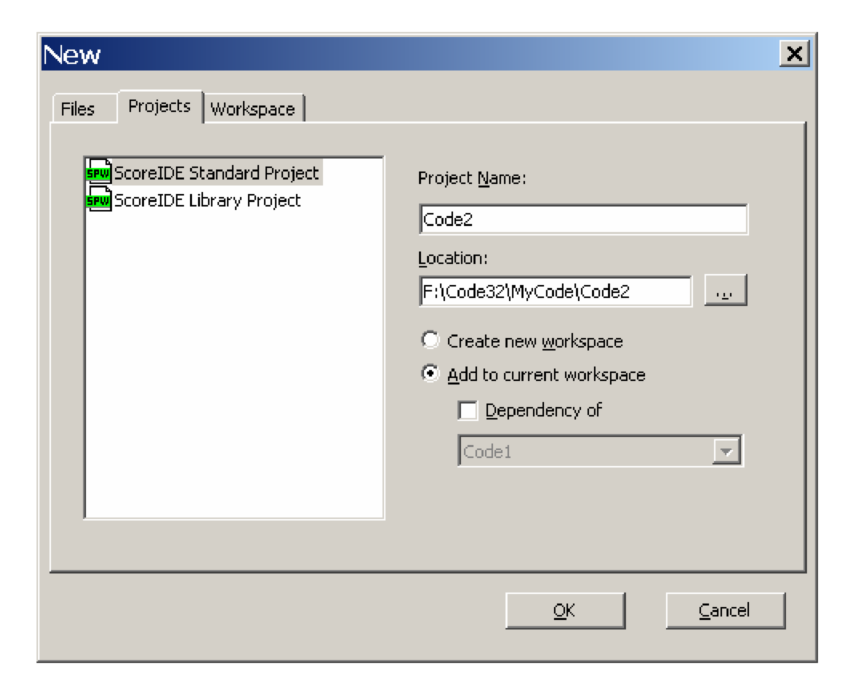


图 5.52 新建工程 Code2

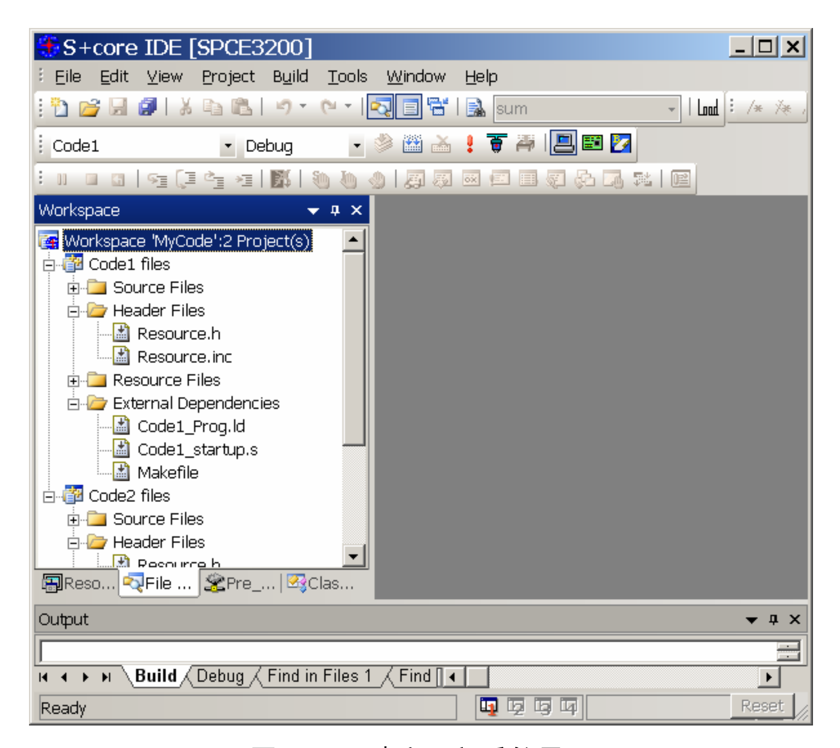


图 5.53 建立工程后的界面

第三步：点击图标选择 Body，Code1 和 Code2 的 Body 都选择为 SPCE3200，如图 5.54。

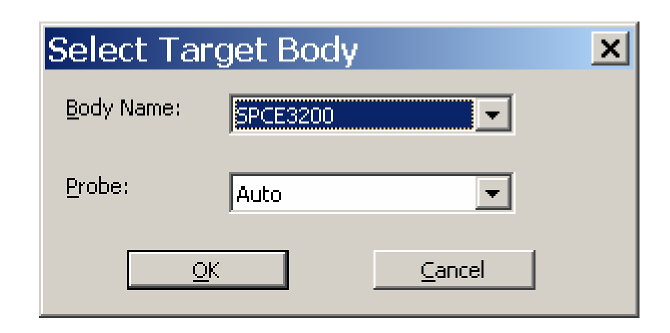


图 5.54 Body 选择为 SPCE3200

第四步：在工程 Code1 和工程 Code2 里各建一个新文件 main.c。

（1）在 Code1 里新建一个 main.c 文件，如图 5.55。注意 Add to Project 选择 Code1。

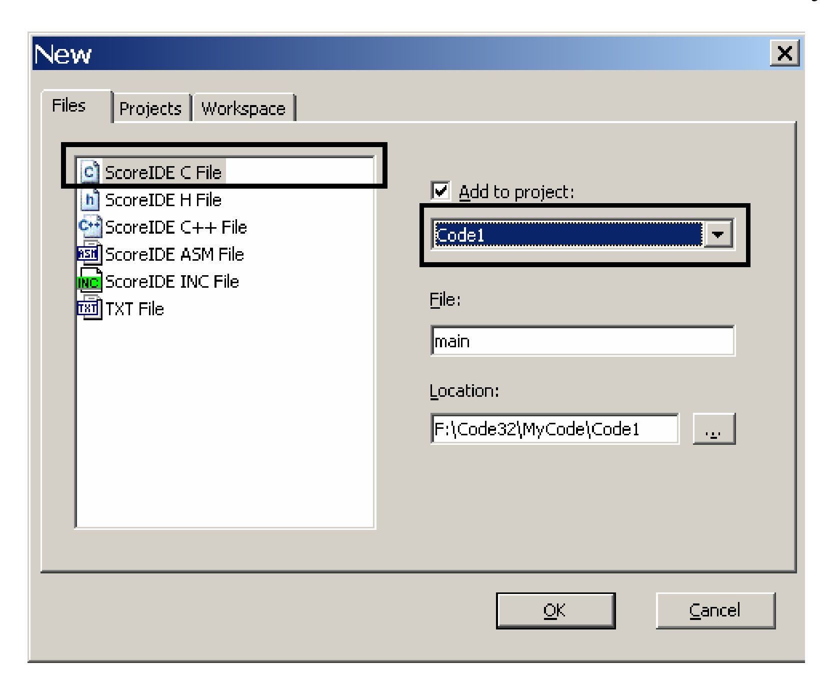


图 5.55 在 Code1 工程里建立文件 main.c

（2）在 Code2 里新建一个 main.c 文件，如图 5.56。注意 Add to Project 选择 Code2。

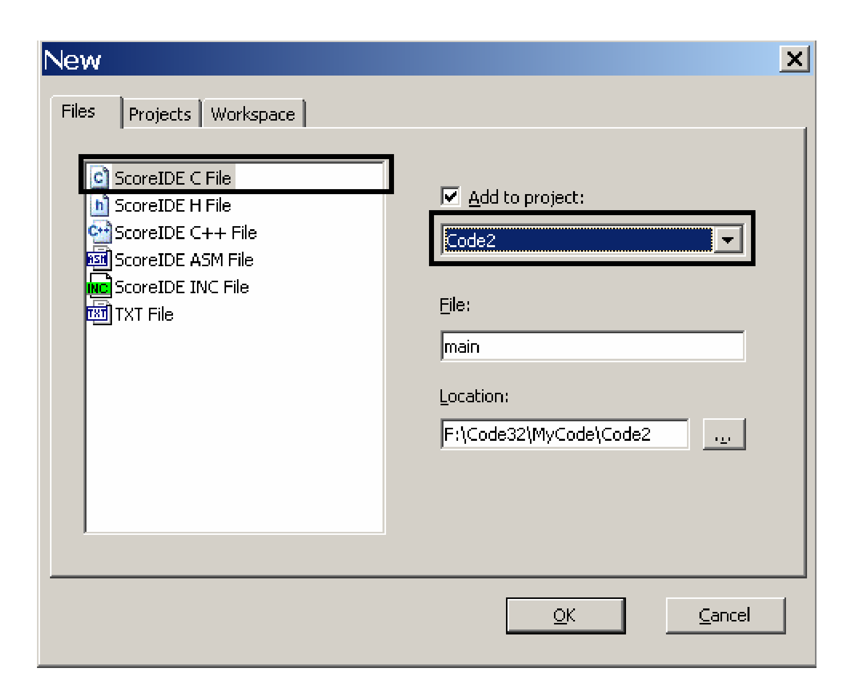


图 5.56 在 Code2 工程里建立文件 main.c

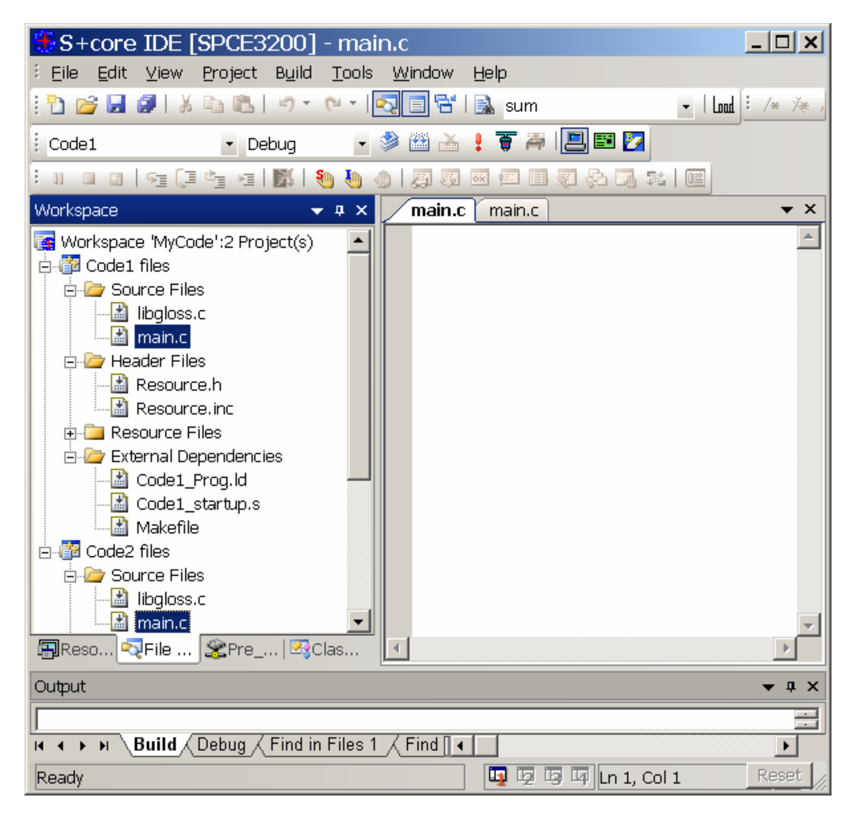


图 5.57 建立文件后的界面

第五步：在 Code1 的 main.c 文件中编写 1 到 100 的累加程序，在 Code2 的 main.c 文件中编写 LED1 灯闪烁的程序。

Code1 的 main.c 程序段：累加结果保存在 sum 中。

```
int main(void)
```

```
{
    int i,sum=0;
    for(i=0;i<=100;i++)
    {
        sum += i;
    }
    while(1);
    return 0;
}
```

Code2 的 main.c 程序段：

```
#define P_IOB_GPIO_SETUP (volatile unsigned int*)0x8820004C
int main(void)
{
    int i;
    *P_IOB_GPIO_SETUP = 0x00002000;
    while(1)
    {
        for(i=0;i<0x0027ffff;i++);
            *P_IOB_GPIO_SETUP ^= 0x00000020;
    }
    return 0;
}
```

第六步：在工具栏里选择 Code1 工程，如图 5.58，选择软件仿真；按 F7 或者点击编译链接生成可执行文件，通过 Output 窗口观察链接信息；按 F8 或者点击下载程序。

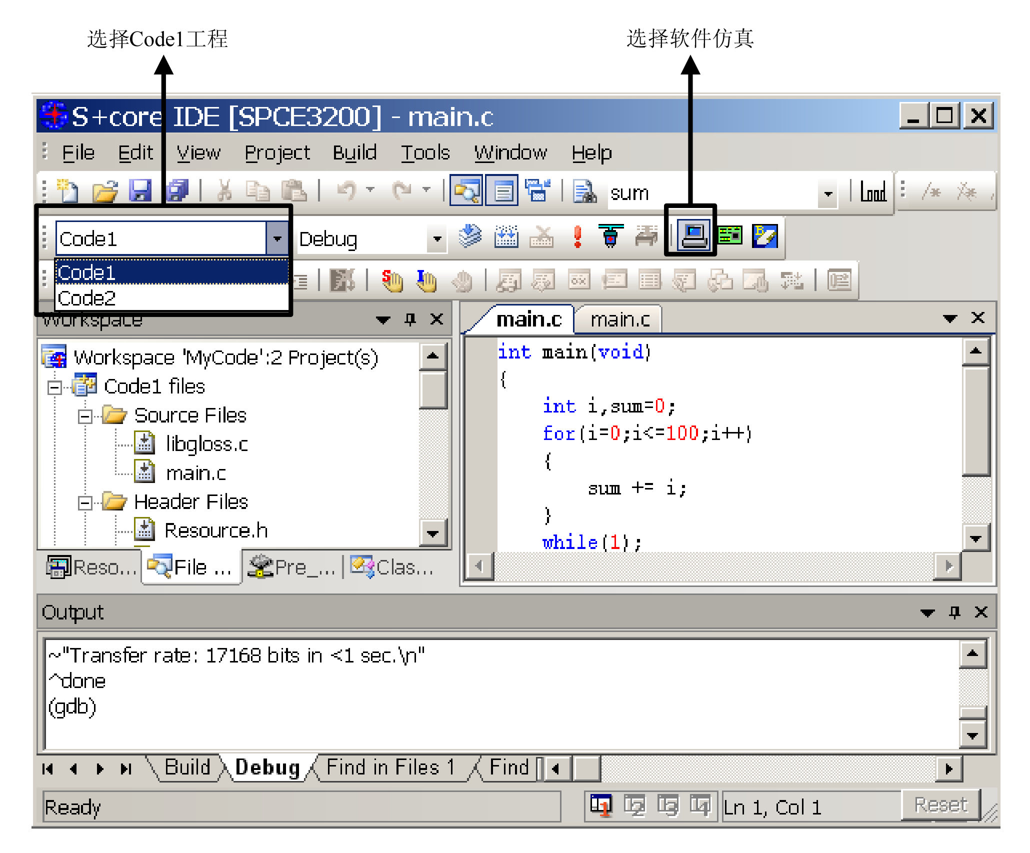


图 5.58 选择 Code1 工程

第七步：按 Alt+6 快捷键或者点击图标打开 Watch 窗口，在 Name 栏依次输入 i、sum；按 Alt+L 快捷键或者点击图标打开 Local Variable 窗口；按 Alt+7 快捷键或者点击图标打开 Registers 窗口。

第八步：如图 5.59，按 F11 或者点击图标单步运行，观察 Watch 窗口和 Local Variable 窗口 i 和 sum 的变化；观察状态栏里 Cycles 的变化。

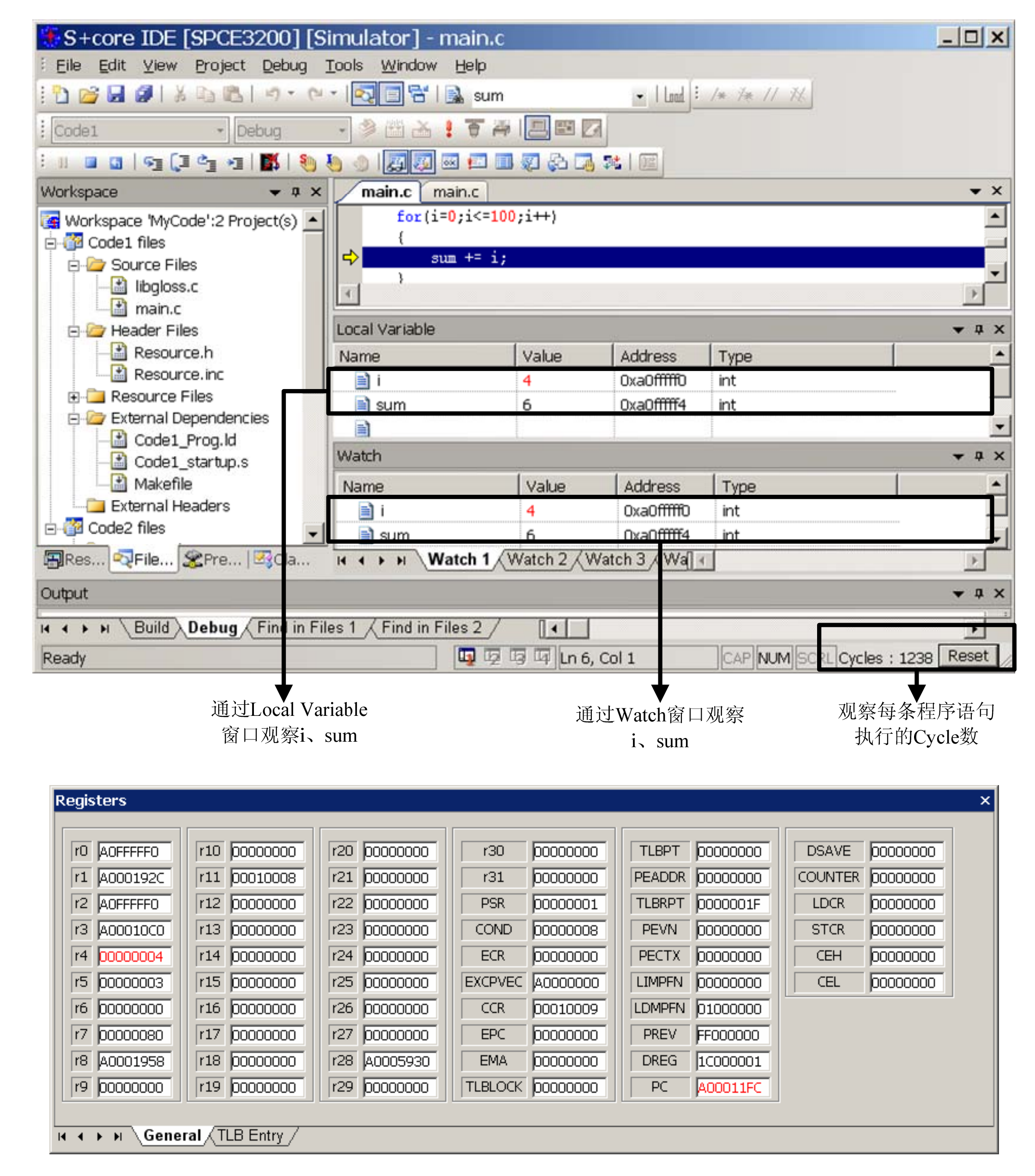
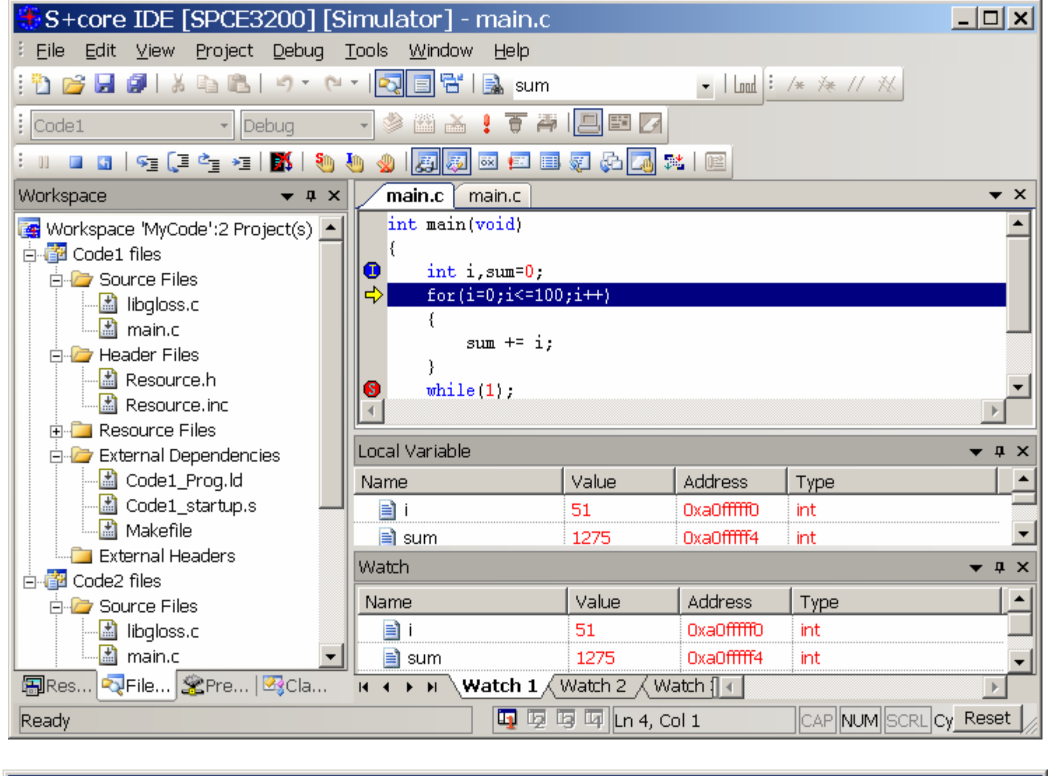


图 5.59 调试界面

第九步：如图 5.60，按 F9 或者点击图标在 while 语句加一个软件断点；按 F6 或者点击图标在第一语句加一个指令断点；按 Alt+B 或者点击图标打开 Breakpoints 窗口，如图 5.60，在 Variable 栏填写变量 i，再在 Condition 栏填写 i= =51；全速运行，观察变量观察窗口此时 sum 的值。



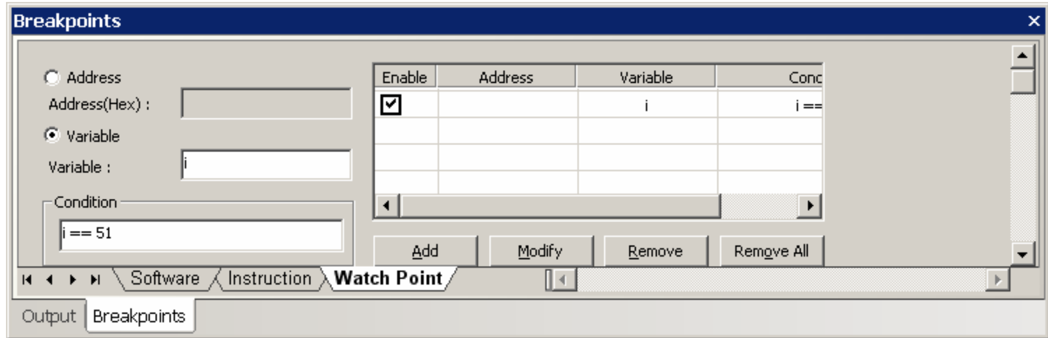


图 5.60 添加断点界面

第十步：如图 5.2 连接硬件；如图 5.61 连接 JP2 中 GPIO2 与 LED1；SPCE3200 的所有跳线和配置开关均为默认设置。

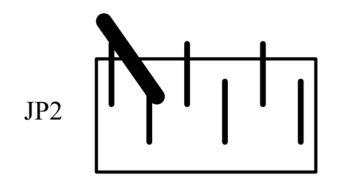


图 5.61 GPIO2 与 LED1 连接

第十一步： 同以上步骤类似的方法，选择 Code2 工程，选择 ICE。编译链接没有错误后，下载到 SPCE3200 开发板上，打开 Memory 窗口，单步运行，观察 Memory 窗口 0x8820004C~0x8820004F 地址单元的数据变化；全速运行，观察 LED1 的现象。

6 SPCE3200 应用实例

SPCE3200 在多媒体影像处理设备、手持 PDA 设备等领域应用优势明显，下面举一个基础应用范例，通过这个例子，读者可以对 SPCE3200 的应用及开发过程有所了解。

6.1 原理概述

本例所设计的是一个通过 UART 远程控制的具有时间和闹铃功能的电子记事本系统。PC 端通过 UART 串行通信总线向 SPCE3200 发送命令，SPCE3200 接收命令并完成对内部的时间、闹铃和日记系统的控制。

PC 端通过串口发送的命令支持如下几种：

- time [年-月-日 时:分:秒]
- alarm [时:分] | [on|off|stop]
- diary [–w|-d]

time 命令可以使 SPCE3200 将其内部时间通过 UART 传送给 PC 端显示，当 time 命令后面带有日期和时间做为参数时，SPCE3200 可以读取这些参数，并根据参数调整其内部时钟和日历系统。

alarm 指令可以查看 SPCE3200 内部的闹钟系统的状态;如果 alarm 指令后面带有时间做为参数，则 SPCE3200 可以读取这些参数，并更改内部的闹钟设置。当闹钟时间到时，SPCE3200 控制 LED 灯闪烁，以提醒用户；alarm on 命令可以打开闹钟功能;alarm off 命令可以关闭闹铃功能;alarm stop 命令可以停止正在闪烁的 LED。

diary 指令可以使 SPCE3200 将非易失性存储器中的一段文本发送给 PC 端显示；如果后面带有“-w”参数，系统允许用户输入一段文字，并将这些文字存储到非易失性存储器中；如果带有“-d”参数，则可以将之前写入的文字内容删除。

6.2 应用分析

分析本例的设计要求，可以知道，使用 SPCE3200 内部的 UART 控制器，可以完成与 PC 机的通信和交互，这部分也是整个系统的基础，其他所有功能，包括接收并解析 PC 端发送过来的命令、向 PC 机回传信息灯，都是以 UART 为基础的，所以，在编写程序时需要首先初始化 UART 控制器，以便搭建整个系统的基础。

时钟功能可以利用 SPCE3200 内部的 RTC 模块来实现。但是由于 RTC 电路只能产生半秒、秒、分、时等信号，不能完成对日期的记录和计算，所以，同时需要通过软件编程来实现日历功能。

SPCE3200 内部的 RTC 模块内置了一个时间比较功能，使用该功能，即可完成闹钟功能的实现。

根据设计要求，系统需要具备将文本内容保存到非易失性存储器，并可以重新读出然后发送给 PC 机的功能，这里，选用 Nor 型 Flash 存储器做为保存文本内容的载体，所以，在该系统中需要对 Nor 型 Flash 进行编程。

使用上面提到的几个模块：UART、RTC、Nor 型 Flash，就可以完成本例要求的功能。

6.3 硬件电路

硬件电路由 LED 控制电路、RS232 电平转换电路构成。

系统通过使用 GPIO 外接三个 LED 灯做为闹钟提示显示，电路如图 4.86 所示。IOB3～IOB5 三个 GPIO 通过电阻 R 接三个 LED 发光二极管，到闹钟时间时，则使发光二极管闪烁示意。

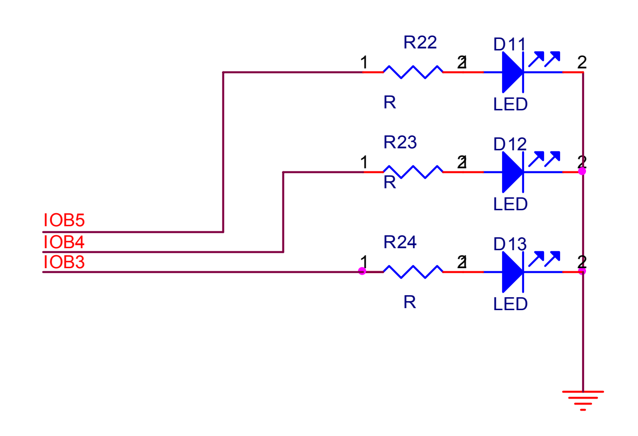


图 6.1 LED 控制电路原理图

RS232 电平转换电路由一片 MAX3232 及其典型应用电路构成，完成 TTL 电平向 232 电平的转换，以便可以直接和 PC 的串口相连。电路如图 6.2 所示。

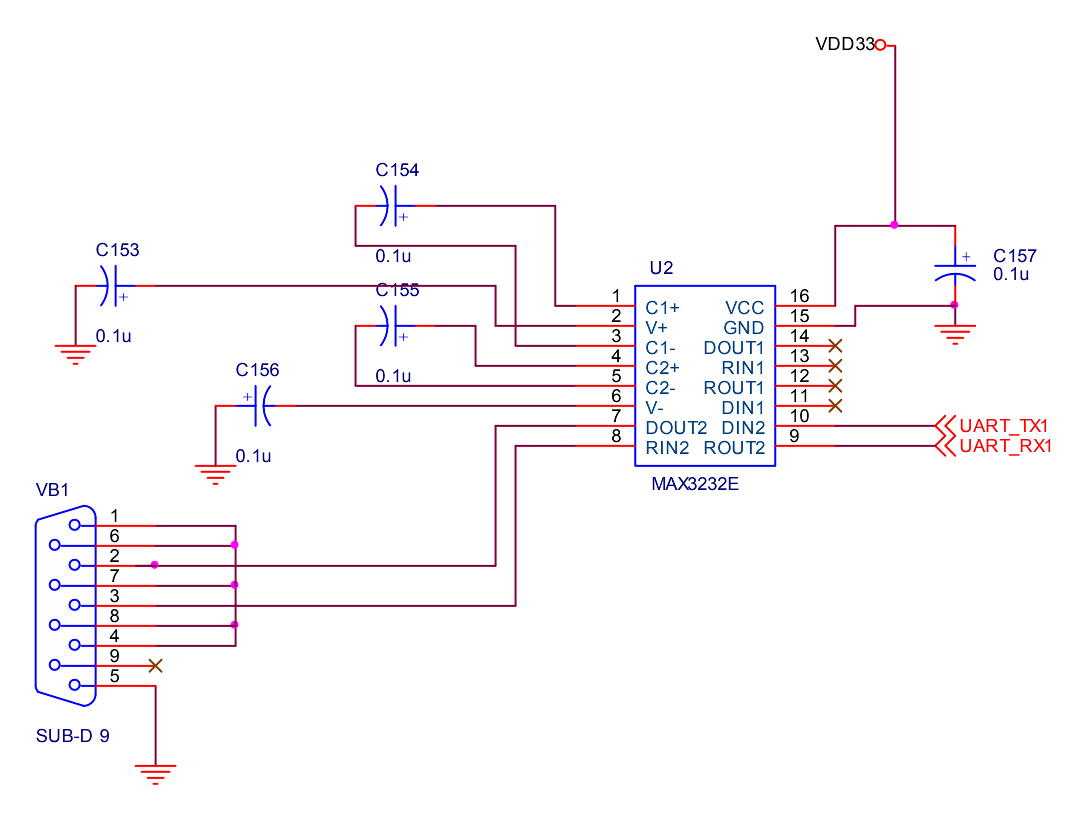


图 6.2 RS232 电平转换电路原理图

用户在使用 SPCE3200 开发板调试时，可以将 JP2 的三组跳线短接，并将 SW3 拨码开关的 7 和 8 两位置为打开状态，然后使用串口线将开发板的串口与 PC 机相连，即可完成硬件连接。

6.4 程序设计

整个程序分为主程序、软件 FIFO 管理程序、UART 收发程序、RTC 控制及日期计算程序、Nor 型 Flash 操作程序、命令获取和分配程序、命令处理程序等。

6.4.1 主程序

程序按照模块化程序设计，所有功能均通过调用子函数实现，主函数比较简单，主要完成系统

初始化工作，然后便等待接收 PC 端命令并做处理。其流程图如图 6.3 所示。

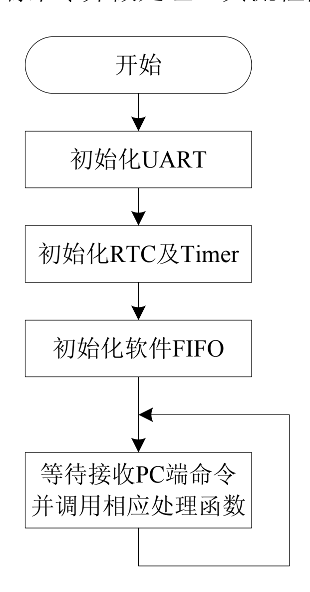


图 6.3 主程序流程图

6.4.2 软件 FIFO 管理程序

PC 端传递过来的命令通过 UART 串行接收，此时要求在接收到一定数量的数据之后判断当前是否是一条有效的命令，而 UART 自带的缓冲 FIFO 不足以缓存一条命令，所以，有必要使用软件编程实现一个 FIFO 缓冲区，用来暂时保存 UART 接收到的数据，这样可以方便命令接收程序判断当前是否已经是一条有效的命令。

软件 FIFO 的基本原理是，首先事先在存储空间中分配一定数量的存储单元，然后定义两个位置计数器，以便记录当前缓冲区内有效数据的范围。当向缓冲区填入或者从缓冲区取出数据时，同时移动位置计数器，并判断是否溢出或清空。

关于软件 FIFO 的具体实现请参考应用例配套代码，这里不再细述。在下面的编程中直接调用软件 FIFO 模块的 API 接口函数即可，常用函数如表 6.1 所示。

表 6.1 主程序流程图

FIFO_Init()	
语法格式	void FIFO_Init(void)
参数	无
返回值	无
功能	初始化软件 FIFO
FIFO_CheckEmpty()	
语法格式	int FIFO_CheckEmpty(void)
参数	无
返回值	0 : FIFO 非空 1 : FIFO 空

功能	检查 FIFO 是否为空
FIFO_Pop()	
语法格式	char FIFO_Pop(void)
参数	无
返回值	FIFO 队首元素
功能	得到 FIFO 队首数据，并将其在 FIFO 中删除
FIFO_Push()	
语法格式	void FIFO_Push(char item)
参数	char item： 待填入的数据
返回值	无
功能	向 FIFO 中填入一个数据

6.4.3 UART 收发程序

在前面的章节中，介绍了 SPCE3200 内部的 UART 收发器的使用方法。在本例中，将使用中断方式来接收 PC 端的数据，并使用查询方式向 PC 机发送反馈信息。

在 UART 的接收中断中，需要将接收到的数据保存至软件 FIFO，以便于此后的处理，如图 6.4 所示。

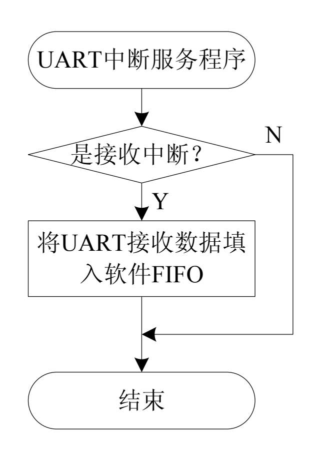


图 6.4 UART 中断接收程序流程图

在后面的程序中可以从 FIFO 中将数据取出，并判断接收到的命令是否有效。

使用查询方式发送数据的代码如下所示。

```
void out(unsigned char ch)
{
    while((*P_UART_TXRX_STATUS & 0x4000) == 0);
    *P_UART_TXRX_DATA = ch;
}
```

6.4.4 RTC 控制及日期计算程序

该部分主要完成时间日期的计算和管理，以及闹钟功能的实现。

SPCE3200 内部的 RTC 模块可以实现时-分-秒的计时和计算，用户只需要设置与三个时间量相关的三个寄存器（P_RTC_TIME_HOUR、P_RTC_TIME_MIN、P_RTC_TIME_SEC），即可为系统设置好初始时间；读取这三个寄存器，即可获取当前的系统时间。关于时间的操作相对比较简单，参考代码片段如下所示：

```
typedef struct {                                    // 时间结构体
    char Hour;                                      // 时
    char Minute;                                    // 分
    char Second;                                    // 秒
} TIME;

//==========================================================================
// 语法格式:     void RTC_GetTime(TIME *tp)
// 实现功能:     得到系统时间
// 参数:         tp   -   时间结构体指针
// 返回值:   无
//==========================================================================
void RTC_GetTime(TIME *tp)
{
    tp->Hour = *P_RTC_TIME_HOUR;
    tp->Minute = *P_RTC_TIME_MIN;
    tp->Second = *P_RTC_TIME_SEC;
}

//==========================================================================
// 语法格式:     void RTC_SetTime(TIME *tp)
// 实现功能:     设置系统时间
// 参数:         tp   -   时间结构体指针
// 返回值:   无
//==========================================================================
void RTC_SetTime(TIME *tp)
{
    *P_RTC_TIME_HOUR = tp->Hour;
    *P_RTC_TIME_MIN = tp->Minute;
    *P_RTC_TIME_SEC = tp->Second;
}
```

RTC 模块并没有提供日期计算功能，所以，需要另外编程实现。这里，首先定义了用于存储年月日的变量，然后使用了 RTC 模块提供的小时中断，以便可以在小时数更新的时候判断当前的日期是否需要更新。RTC 中断服务程序中关于日期更新的流程如图 6.5 所示。

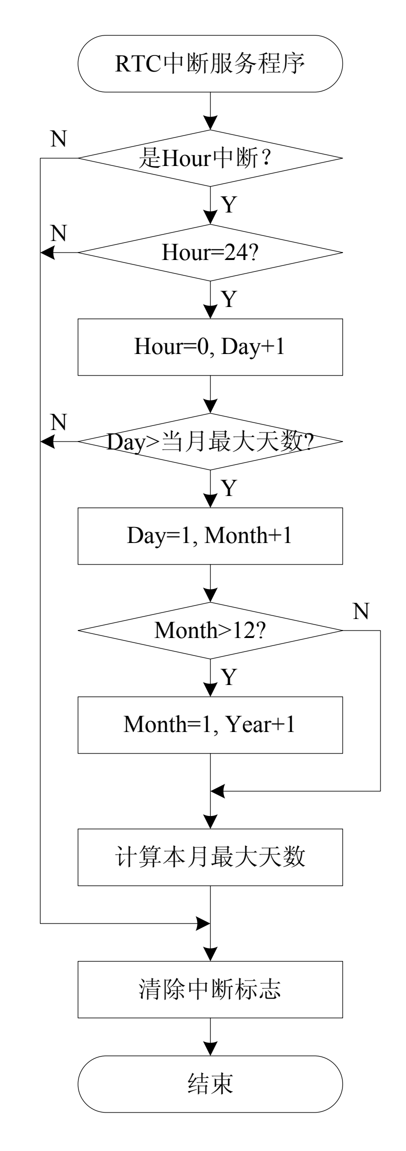


图 6.5 RTC 中断服务程序流程图（1）

在 RTC 模块中同时提供时间比对（Alarm）功能，当 RTC 的时分秒与预设的 Alarm 时间的时分秒相等时，可以触发 Alarm 中断，以便用户可以处理定时事件。这里，将使用它实现闹钟功能。RTC 模块的 Alarm 中断服务程序流程如图 6.6 所示。

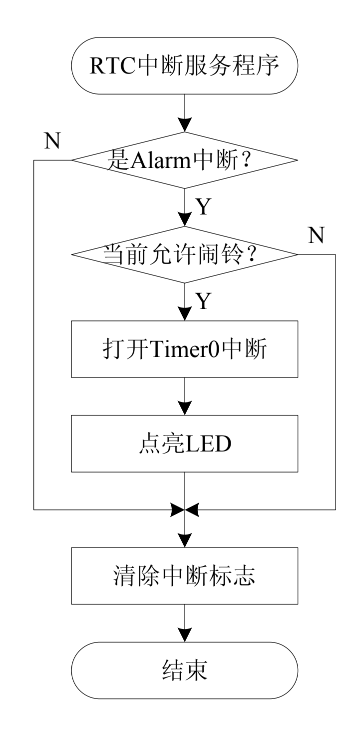


图 6.6 RTC 中断服务程序流程图（2）

其中，Timer0 被初始化为 4Hz 的溢出频率，在 Timer0 的中断服务程序中，使 IOB3～IOB5 的输出数据取反，从而实现 LED 灯的闪烁。同时，在 Timer0 的中断服务程序中对闪烁次数计数，用来控制 LED 闪烁的总时间。Timer0 的中断服务程序流程如图 6.7 所示。

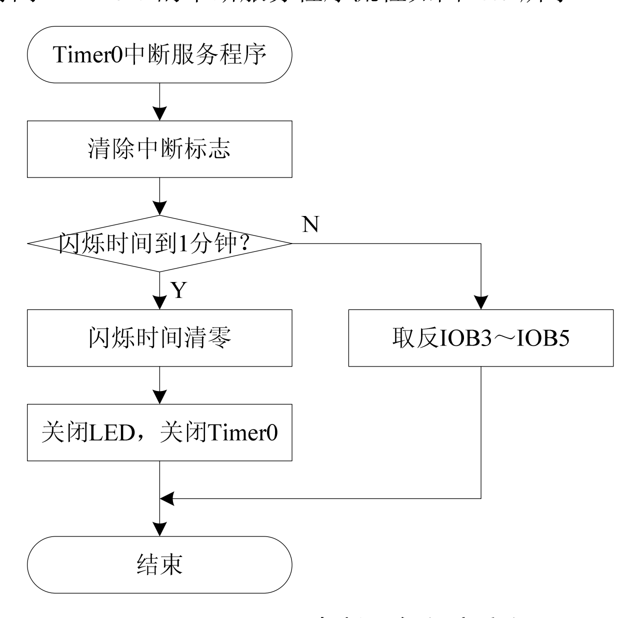


图 6.7 Timer0 中断服务程序流程图

6.4.5 Nor 型 Flash 操作程序

Nor 型 Flash 主要用来保存 PC 端发送记事命令之后发送过来的文本内容。对 Nor 型 Flash 的操作在前面的章节中已经介绍过，这里不再赘述。

6.4.6 命令获取和分配程序

命令获取程序是与 PC 端交互的重要核心。如何识别当前是否是一条有效的命令是程序设计的

关键。在本例中，为了简便起见（同时也可以方便其他扩展），规定从 PC 端发送过来的命令（包括参数）以回车换行符（\r\n）做为结束字符。如，PC 端发送“alarm on”命令，实际通过 UART 发送的数据为：0x61,0x6c,0x61,0x72,0x6d,0x20,0x6f,0x6e,0x0d,0x0a。这样，在程序中就可以以接收到一行字符为基准处理数据。接收一行字符的程序流程如图 6.8 所示。

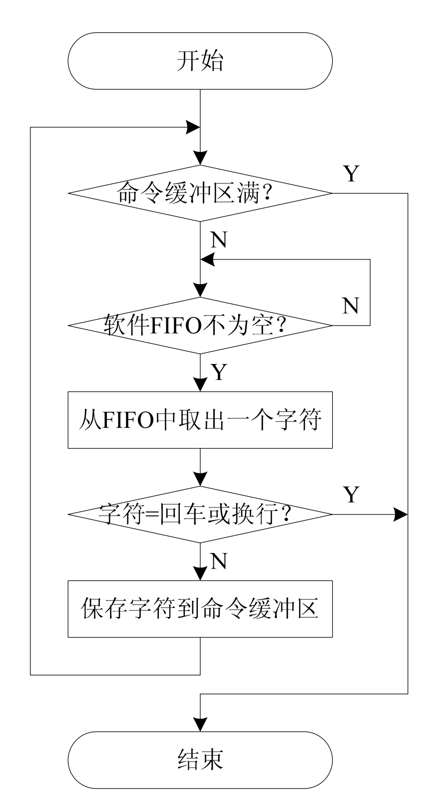


图 6.8 接收一行字符的程序流程图

接收到一行字符后，就可以分析是否包含有有效的命令，并调用相应的处理函数进行处理。程序流程如图 6.9 所示。

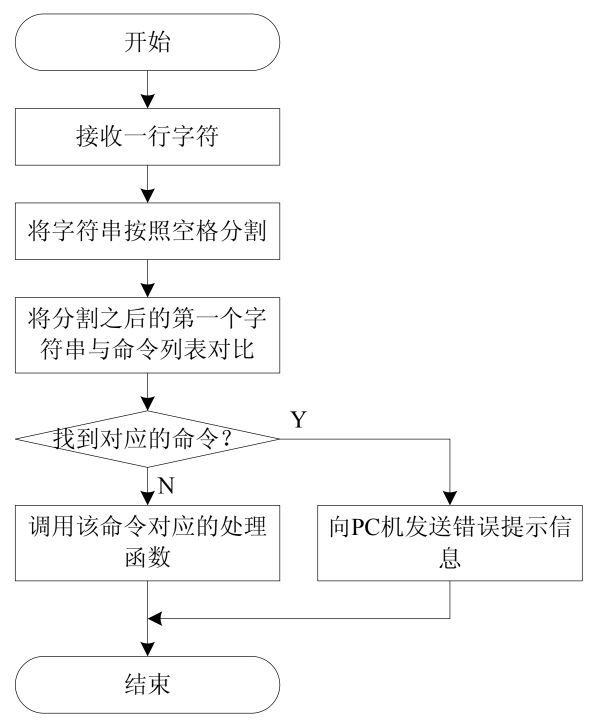


图 6.9 命令解析分配程序流程图

6.4.7 命令处理程序

命令处理程序与系统支持的命令一一对应。按照本例的要求，系统支持“time”、“alarm”、“diary”三条命令，所以，对应的处理程序也有三个：时间命令处理程序、闹钟命令处理程序和记事本命令处理程序。下面分别介绍三个处理程序的实现。

1) 时间命令处理程序

在时间处理程序中，首先需要分析 PC 端发送的“time”命令后面是否带有参数，如果带有参数，则需要将参数分别转换为日期数据和时间数据，并分析日期和时间是否合法，如果合法才能对系统时间进行更新，否则，不能接受给出的设置参数。这部分程序流程如图 6.10 所示。

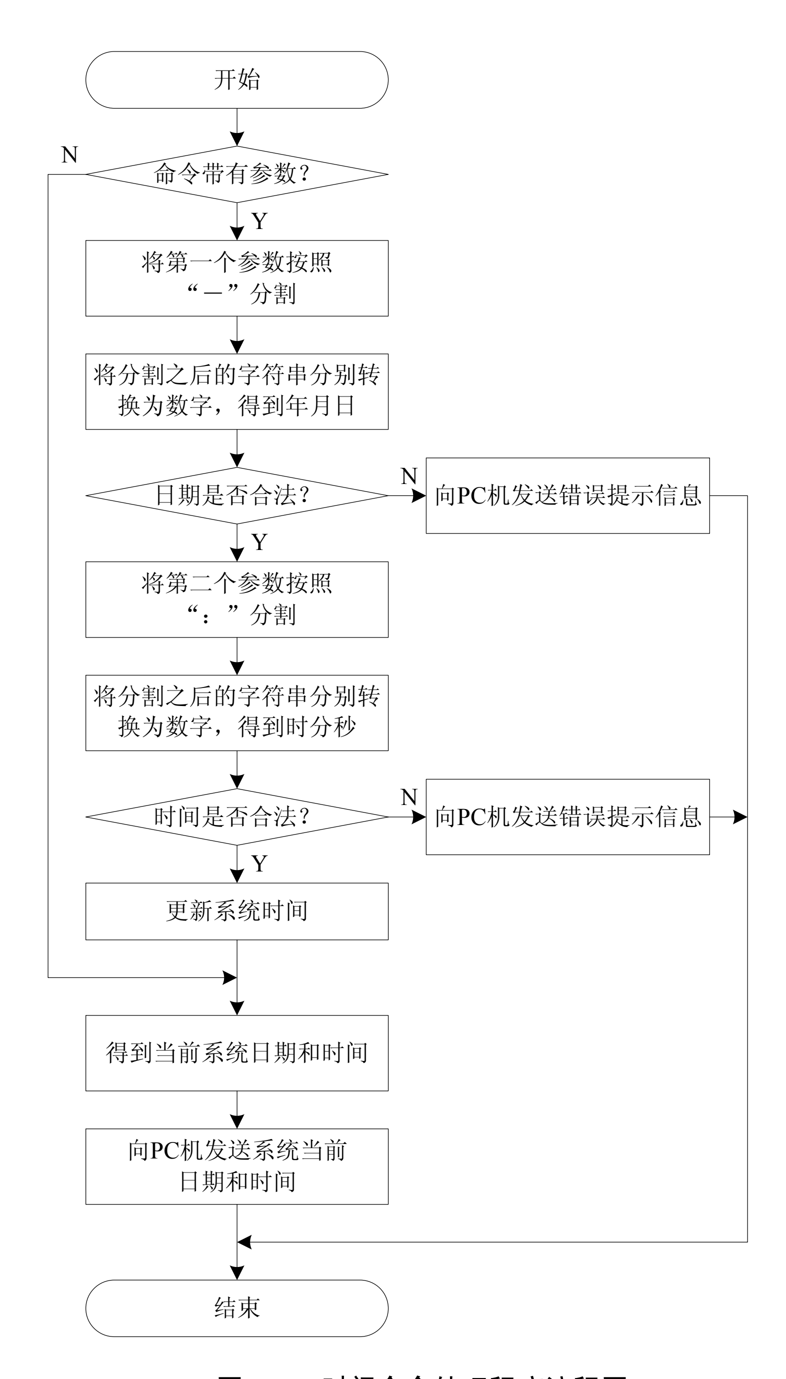


图 6.10 时间命令处理程序流程图

2) 闹钟命令处理程序

闹钟命令可以实现闹钟状态的查询、对闹钟时间的设置、闹铃的打开或关闭等，在这个命令的处理程序中，需要分析命令后面所带的参数的意义，并根据不同的参数做出不同的处理。这部分程序的流程如图 6.11 所示。

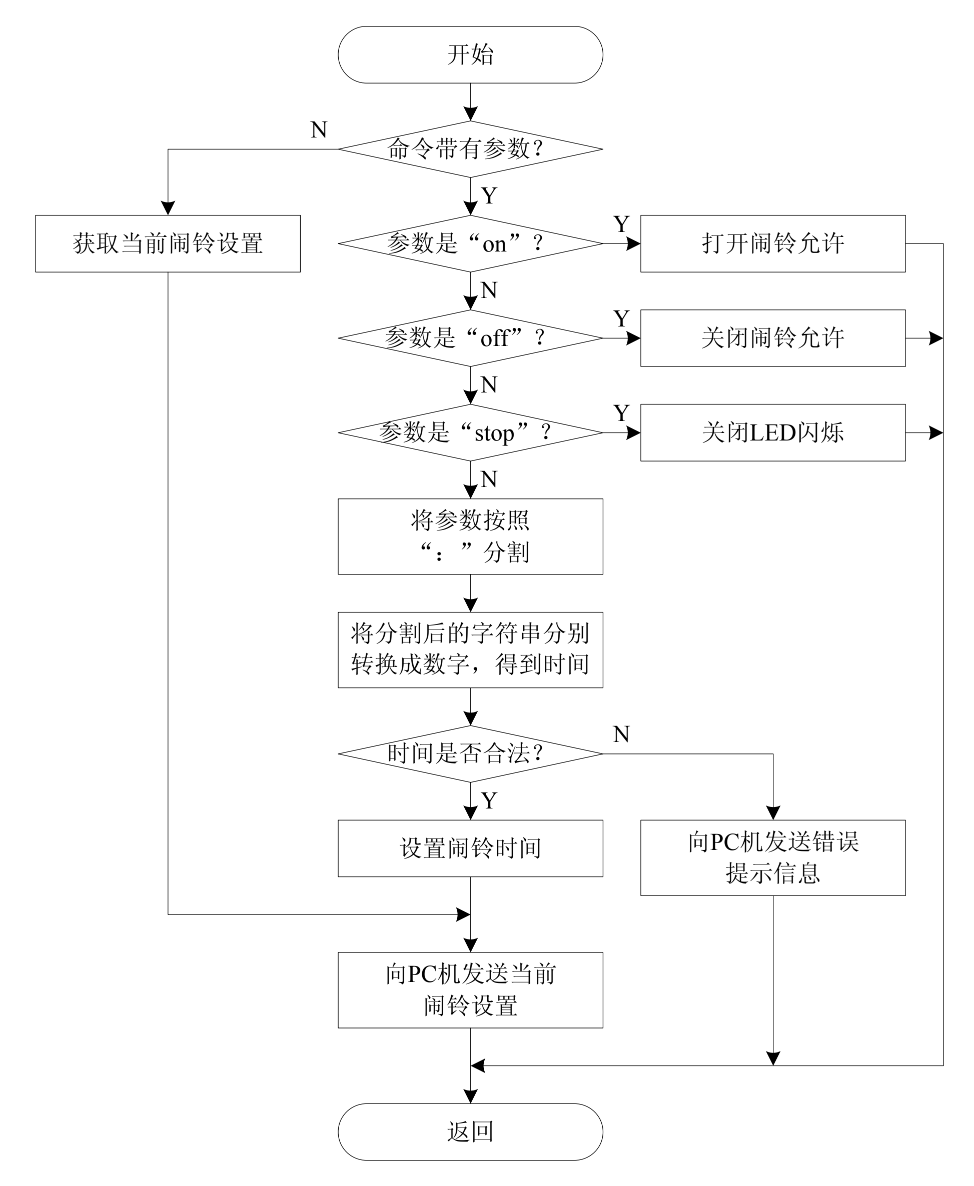


图 6.11 闹钟命令处理程序流程图

3) 记事本命令处理程序

记事本命令处理程序需要完成记事本内容的显示或记事本内容的保存。在本例中，记事本的内容从 Nor 型 Flash 的某个扇区开始记录，为了便于记录文本信息同时简化编程，在本应用例的实现中规定，在扇区的起始 4 个字节中写入“SUN ”表示当前有记事文本记录，并在接下来的 4 个字节中保存文本内容的长度，同时规定，文本文本内容不能超过一个扇区（4KB）大小。

记事本命令处理程序的流程如图 6.12 所示。

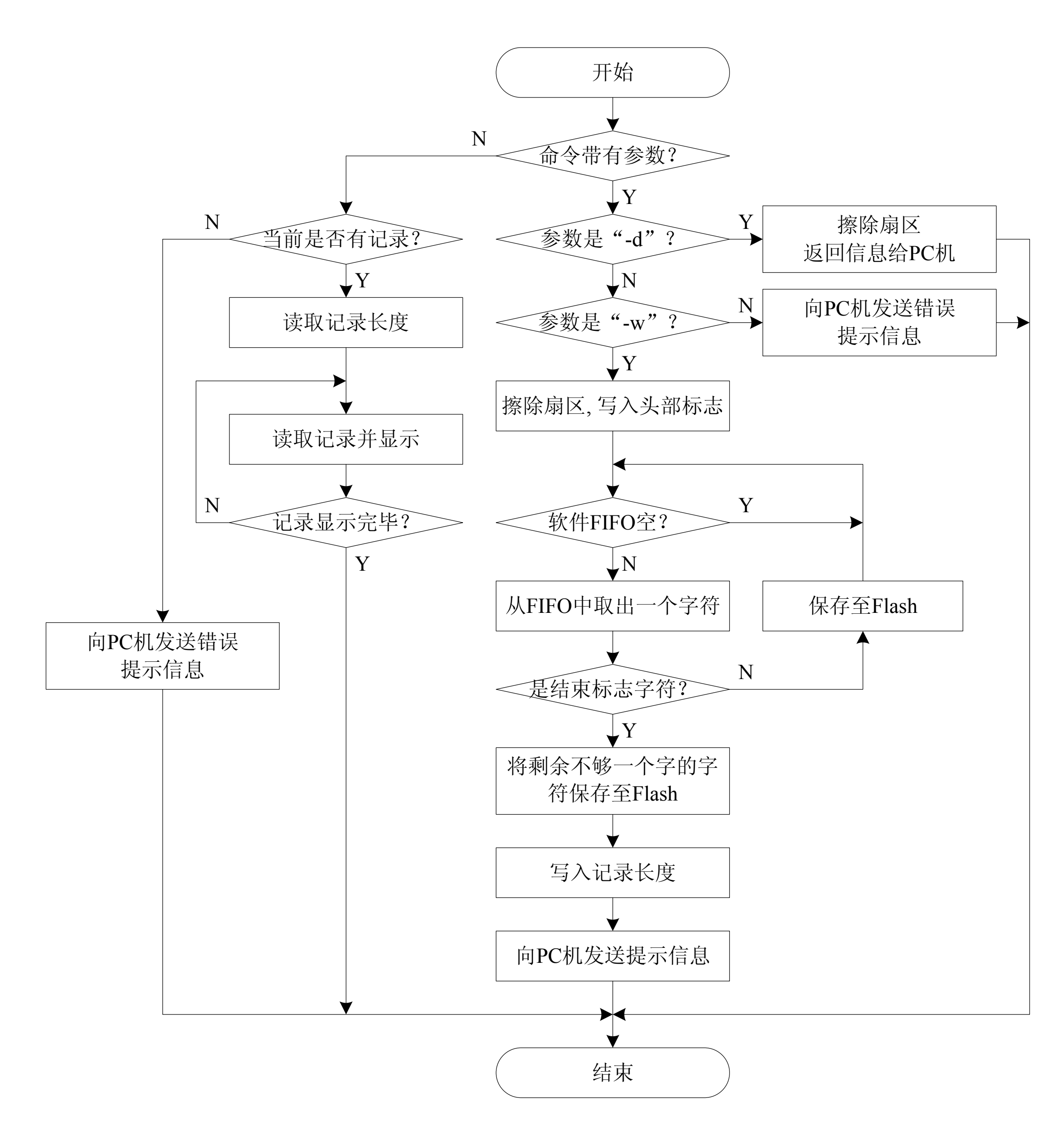


图 6.12 记事本命令处理程序流程图

7 附录

7.1 常用术语、缩写和约定解释

7.1.1 术语

表 7.1 书中用到的术语表

术语	解释
RISC	Reduced Instruction Set Computing 的缩写，精简指令集。
ISA	Instruction Set Architecture 的缩写，指令集构架。 SPCE3200 采用了凌阳自身的指令构架，支持 32 位与 16 位混合指令模式以及并行条件执行。
AMBA	Advanced Microcontroller Bus Architecture 的缩写，增强型微控制器总线构架。 AMBA 协议为设计高性能嵌入式微控制器提供了片上通信的标准，主要包括三种总线：AHB、ASB、APB。
AHB	Advanced High-performance Bus 的缩写，增强型高性能总线。 AHB 是高性能系统的主干总线，主要连接控制器与片上存储器、片外存储器接口或者其他高速低功耗高性能外设单元。
ASB	Advanced System Bus 的缩写，高性能系统总线。 ASB 是介于 AHB 与 APB 之间的一种总线。
APB	Advanced Peripheral Bus 的缩写，增强型外设总线。 APB 主要用来连接一些低功耗外设单元。
SOC	System On a Chip 的缩写，片上系统。
SJTAG	Sunplus JTAG 的缩写。 JTAG 是 Joint Test Action Group 的缩写，最初用来对芯片进行测试，现在大部分用来实现在线编程。
Harvard	哈佛结构，是一种将程序存储器和数据存储器分开的结构形式。
Byte	字节，8 位。
Halfword	半字，16 位。
Word	字，32 位。
NTSC 制/PAL 制	NTSC 是 National Television Standards Committee 的缩写，翻译为（美国）国家电视标准委员会；PAL 是 Phase Alternating Line 的缩写，翻译为逐行倒相。 NTSC 和 PAL 是属于全球两大主要的电视广播制式，其分辨率不同，PAL 制式使用的是 720*576，而 NTSC 制式使用的是 760*480。
I-Cache	Instruction Cache 的缩写，指令 cache。
D-Cache	Data Cache 的缩写，数据 cache 。



DRAM	Dynamic Random Access Memory 的缩写，动态随机存储器。
SDRAM	Synchronous DRAM 的缩写，同步动态随机存储器。
MMU	Memory Management Unit 的缩写，内存管理单元。
LDM	Local Data Memory 的缩写，本地数据存储器。
LIM	Local Instruction Memory 的缩写，本地指令存储器。
DMA	Direct Memory Access 的缩写，直接内存存取。 DMA 是实现高速外设与存储器之间自动成批交换数据而尽量减少 CPU 干预的输入输出操作方式。
BLNDMA	Blending and DMA controller 的缩写，DMA 控制器。
APBDMA	APB Bridge and DMA Controller 的缩写，APB 桥和 DMA 控制器。
FIFO	First In First Out 的缩写，先入先出。是实现数据先入先出的存储器件。
PLL	Phase Locked Logic 的缩写，锁相环逻辑电路。
CKG	Clock Generation 的缩写，时钟产生逻辑。
MIU	Memory Interface Unit 的缩写，存储器接口单元。
BUFCTL	Buffer Controller 的缩写，缓存控制器。
GPIO	General-Port for Input/Output 的缩写，通用输入输出口。
Timer	定时器。
RTC	Real Time Clock 的缩写，实时时钟。
TMB	Time Base 的缩写，时基。
WDOG	Watch Dog 的缩写，看门狗。
ADC	Analog to Digital Converter 的缩写，模/数转换器。
MIC	Microphone 的缩写。
DAC	Digital to Analog Converter 的缩写，数/模转换器。
UART	Universal Asynchronous Receiver/Transmitter 的缩写，通用异步收发传输器。
SPI	Serial Peripheral Interface 的缩写，串行外围接口。
I2C	Inter-Integrated Circuit 的缩写。
SIO	Serial Interface I/O 的缩写，串行输入输出接口。
USB	Universal Serial Bus 的缩写，通用串行总线。
SD	Secure Digital Card 的缩写，安全数码卡，是一种基于半导体快闪记忆器的新一代记忆设备。
TFT	Thin Film Transistor 的缩写，薄膜场效应晶体管，液晶显示器上的每一液晶象素点都是由集成在其后的薄膜晶体管来驱动。

STN　　Super Twisted Nematic 的缩写，超扭曲向列。

ECC　　Error Checking and Correcting 的缩写，错误检查纠正。

CSI　　CMOS Sensor Interface 的缩写，场效应管传感器接口。

JPEG　　Joint Photographic Experts Group 的缩写，联合影像专家组。

MPEG4　　Moving Picture Experts Group 4 的缩写，移动图像专家组 4。

LVR　　Low Voltage Reset 的缩写，低电压复位。

7.1.2　缩写

在 SPCE3200 芯片的寄存器中使用了助记符，有些助记符使用了缩写，表 7.2 列出了缩写的全称及解释：

表 7.2 助记符缩写、全称及解释

序号	缩写	全称	解释
1	CLK	Clock	时钟
2	PLLV	Video PLL	音频锁相环逻辑
3	PLLA	Audio PLL	视频锁相环逻辑
4	PLLU	USB PLL	USB 锁相环逻辑
5	PLLAU	PLLA&PLLU	-
6	32K	32768	32768Hz 晶振
7	CONF	Config	配置
8	ADDR	Address	地址
9	SA	Start Address	起始地址
10	EA	End Address	结束地址
11	BA	Base Address/Buffer A	基地址/缓存 A
12	OA	Offset Address	偏移地址
13	BB	Buffer B	缓存 B
14	CTRL	Control	控制
15	SEL	Select	选择
16	CLR	Clear	清除
17	ERR	Error	错误
18	EN	Enable	使能
19	DIS	Disable	禁能

序号	缩写	全称	解释
20	RST	Reset	复位
21	TX	Transmit	发送
22	RX	Recive	接收
23	TXRX	Transmit& Recive	收发
24	LP	Line Parity	行校验
25	CP	Column Parity	列校验
26	CMP	Compare	比较
27	AINPUT	Analog Input	模拟输入口
28	REQ	Request	请求
29	PRI	Priority	优先级
30	KEYC	Key Change	键值改变，主要用于键唤醒
31	FMT	Format	格式
32	HOR	Horizontal	水平
33	SYNC	Synchronization	同步
34	VER	Vertical	垂直
35	ACT	Active	激活
36	COL	Column	列
37	SEQ	Sequence	顺序
38	NUM	Number	数字
39	CCP	Capture/Comparison/PWM	捕获/比较/脉宽调制输出
40	SEC	Second	秒
41	MIN	Minute/Minimal	分/最小
42	ALM	Alarm	报警
43	INT	Interrupt	中断
44	NAND	Nand Flash	Nand 型 Flash

7.1.3 约定

在 SPCE3200 芯片的寄存器中使用了助记符，对助记符的字段进行了约定，列出了这些字段及解释：

表 7.3 助记符的一些约定

序号	英文	约定
1	CONF	Config，配置：主要用于一个时钟的打开/关闭配置。
2	STATUS	状态：对于一些既有使能又有状态或者只有状态的描述。
3	CTRL	Control，控制：模块使能或者模式的选择控制。
4	SEL	Select，选择：主要是对于给定的一些条件的选择，比如频率或者接口的选择。
5	SETUP	设置：用于一些频率的设置和 GPIO 的设置。
6	INPUT	输入数据。
7	ADDR	地址。
8	DATA	数据。
9	PULL	上拉或者下拉。
10	COMMAND	命令，给一个寄存器写一个具体的命令。
11	MODE	模式，如输入输出模式等。
12	INTERFACE	接口，主要是一些复用接口。
13	INT	中断，包括模块的中断、中断控制器等。
14	CLR	清除，一般在用在只写的寄存器。

7.2　CPU 内核寄存器速查表

序号	寄存器名	内核寄存器功能	函数调用时是否被保存
1	r0	通用寄存器 0 用作堆栈指针	是
2	r1	通用寄存器 1 用作暂存器，通常用于汇编器	否
3	r2	通用寄存器 2 用作帧（页）指针	是
4	r3	通用寄存器 3 用作链接寄存器。函数调用时，此寄存器保存返回地址	是
5	r4	通用寄存器 4 用作用于传递第 1 个参数及返回值	否
6	r5	通用寄存器 5 用作用于传递第 2 个参数	否
7	r6	通用寄存器 6 用作用于传递第 3 个参数	否
8	r7	通用寄存器 7 用作用于传递第 4 个参数	否
9	r8	通用寄存器 8 用作调用者保存的寄存器	否
10	r9	通用寄存器 9 用作调用者保存的寄存器	否

序号	寄存器名	内核寄存器功能	函数调用时是否被保存
11	r10	通用寄存器 10 用作调用者保存的寄存器	否
12	r11	通用寄存器 11 用作调用者保存的寄存器	否
13	r12	通用寄存器 12 用作被调用者保存的寄存器，或选择用于保存帧指针	是
14	r13	通用寄存器 13 用作被调用者保存的寄存器，或选择用于保存帧指针	是
15	r14	通用寄存器 14 用作被调用者保存的寄存器，或选择用于保存帧指针	是
16	r15	通用寄存器 15 用作被调用者保存的寄存器，或选择用于保存帧指针	是
17	r16	通用寄存器 16 用作被调用者保存的寄存器，或选择用于保存帧指针	是
18	r17	通用寄存器 17 用作被调用者保存的寄存器，或选择用于保存帧指针	是
19	r18	通用寄存器 18 用作被调用者保存的寄存器，或选择用于保存帧指针	是
20	r19	通用寄存器 19 用作被调用者保存的寄存器，或选择用于保存帧指针	是
21	r20	通用寄存器 20 用作被调用者保存的寄存器，或选择用于保存帧指针	是
22	r21	通用寄存器 21 用作被调用者保存的寄存器，或选择用于保存帧指针	是
23	r22	通用寄存器 22 用作调用者保存的寄存器	否
24	r23	通用寄存器 23 用作调用者保存的寄存器	否
25	r24	通用寄存器 24 用作调用者保存的寄存器	否
26	r25	通用寄存器 25 用作调用者保存的寄存器	否
27	r26	通用寄存器 26 用作调用者保存的寄存器	否
28	r27	通用寄存器 27 用作调用者保存的寄存器	否
29	r28	通用寄存器 28 用作全局指针	是
30	r29	通用寄存器 29，编译器保留	是
31	r30	通用寄存器 30 仅为操作系统使用	是
32	r31	通用寄存器 31 仅为操作系统使用	是

序号	寄存器名	内核寄存器功能	函数调用时是否被保存
33	CEH	乘/除法运算特殊寄存器，保存乘法运算结果的高 32 位或除法结果的余数	否
34	CEL	乘/除法运算特殊寄存器，保存乘法运算结果的低 32 位或除法结果的商	否
35	Sr0(CNT)	特殊功能寄存器 0 用于计数器操作	否
36	Sr1(LCR)	特殊功能寄存器 1 用于装载合并指令操作	否
37	Sr2(SCR)	特殊功能寄存器 2 用于存储合并指令操作	否
38	Cr0(PSR)	程序状态寄存器，主要用来进行异常使能，Endian 的选择等。	-
39	Cr1(Condition)	条件寄存器，N、Z、C 等条件标志位。	-
40	Cr2(ECR)	异常原因寄存器	-
41	Cr3(EXCPVec)	异常向量寄存器	-
42	Cr4(CCR)	Cache 控制寄存器	-
43	Cr5(EPC)	异常程序计数器	-
44	Cr6(EMA)	异常存储器地址寄存器	-
45	Cr7(TLBLock)	TLB 时钟寄存器	-
46	Cr8(TLBPT)	TLB 指针寄存器	-
47	Cr9(PEADDR)	页入口地址寄存器	-
48	Cr10(TLBRPT)	TLB 随机指针寄存器	-
49	Cr11(PEVN)	页入口的实际页数寄存器	-
50	Cr12(PECTX)	页入口内容寄存器	-
51	Cr13	-	-
52	Cr14	-	-
53	Cr15	LIM 物理帧号	-
54	Cr16	LDM 物理帧号寄存器	-
55	Cr17	-	-
56	Cr18(Prev)	Prev 寄存器，保存处理器的版本及修订信息	-
57	Cr29(DREG)	DREG 寄存器，保留 Debug 的一些信息	-
58	Cr30(DEPC)	Debug 异常程序计数器寄存器	-
59	Cr31(DSAVE)	Debug 异常内容保存寄存器	-

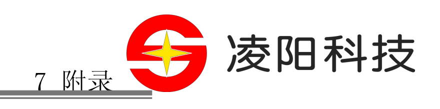


7.3 硬件模块寄存器速查表

表 7.4 硬件模块寄存器速查表

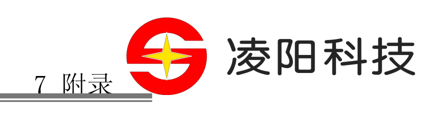
序号	寄存器名称	助记符	地址
时钟模块			
1	CPU 时钟选择寄存器	P_CLK_CPU_SEL	0x88210004
2	AHB 总线时钟配置寄存器	P_CLK_AHB_CONF	0x88210008
3	AHB 总线时钟选择寄存器	P_CLK_AHB_SEL	0x8821000C
4	PLLV 时钟配置寄存器	P_CLK_PLLV_CONF	0x882100B4
5	PLLV 时钟选择寄存器	P_CLK_PLLV_SEL	0x882100B8
6	PLLAU 时钟配置寄存器	P_CLK_PLLAU_CONF	0x882100bc
7	32K 实时时钟配置寄存器	P_CLK_32K_CONF	0x88210114
8	PLLV 稳定输出时间寄存器	P_PLLV_STABLE_TIME	0x88210104
中断控制器模块			
9	中断时钟配置寄存器	P_INT_CLK_CONF	0x882100D0
10	中断请求状态寄存器 1	P_INT_REQ_STATUS1	0x880a0000
11	中断请求状态寄存器 2	P_INT_REQ_STATUS2	0x880a0004
12	中断优先级寄存器	P_INT_GROUP_PRI	0x880a0008
13	第 0 组中断优先级寄存器	P_INT_GROUP0_PRI	0x880a0010
14	第 1 组中断优先级寄存器	P_INT_GROUP1_PRI	0x880a0014
15	第 2 组中断优先级寄存器	P_INT_GROUP2_PRI	0x880a0018
16	第 3 组中断优先级寄存器	P_INT_GROUP3_PRI	0x880a001C
17	中断控制寄存器 1	P_INT_MASK_CTRL1	0x880a0020
18	中断控制寄存器 2	P_INT_MASK_CTRL2	0x880a0024
MIU 模块			
19	MIU 时钟配置寄存器	P_MIU_CLK_CONF	0x88210010
20	SDRAM 电源寄存器	P_MIU_SDRAM_POWER	0x8807005C
21	SDRAM 参数设置寄存器 1	P_MIU_SDRAM_SETUP1	0x88070060
22	SDRAM 状态寄存器	P_MIU_SDRAM_STATUS	0x8807006C
23	SDRAM 参数设置寄存器 2	P_MIU_SDRAM_SETUP2	0x88070094
BUFFER CONTROL 模块			

序号	寄存器名称	助记符	地址
24	BUFCTRL 时钟配置寄存器	P_BUFCTRL_CLK_CONF	0x882100C8
25	BUFCTRL 控制寄存器 1	P_BUFCTRL_MODE_CTRL1	0x88090000
26	BUFCTRL 缓冲区选择寄存器	P_BUFCTRL_BUFFER_SEL	0x88090004
27	BUFCTRL 中断控制寄存器	P_BUFCTRL_INT_CTRL	0x8809004C
28	BUFCTRL 帧丢失中断控制寄存器	P_BUFCTRL_FRAMELOSS_INT	0x88090028
29	BUFCTRL 控制寄存器 2	P_BUFCTRL_MODE_CTRL2	0x8809003C
30	BUFCTRL 控制寄存器 3	P_BUFCTRL_MODE_CTRL3	0x88090040
APB DMA 模块			
31	DMA 时钟配置及存器	P_DMA_CLK_CONF	0x88210058
32	DMA 忙状态寄存器	P_DMA_BUSY_STATUS	0x88080000
33	DMA 中断状态寄存器	P_DMA_INT_STATUS	0x88080004
34	AHB DMA 第 0 通道缓冲区 A 起始地址	P_DMA_AHB_SA0BA	0x88080008
35	AHB DMA 第 1 通道缓冲区 A 起始地址	P_DMA_AHB_SA1BA	0x8808000C
36	AHB DMA 第 2 通道缓冲区 A 起始地址	P_DMA_AHB_SA2BA	0x88080010
37	AHB DMA 第 3 通道缓冲区 A 起始地址	P_DMA_AHB_SA3BA	0x88080014
38	AHB DMA 第 0 通道缓冲区 A 结束地址	P_DMA_AHB_EA0BA	0x88080018
39	AHB DMA 第 1 通道缓冲区 A 结束地址	P_DMA_AHB_EA1BA	0x8808001C
40	AHB DMA 第 2 通道缓冲区 A 结束地址	P_DMA_AHB_EA2BA	0x88080020
41	AHB DMA 第 3 通道缓冲区 A 结束地址	P_DMA_AHB_EA3BA	0x88080024
42	APB DMA 第 0 通道起始地址	P_DMA_APB_SA0	0x88080028
43	APB DMA 第 1 通道起始地址	P_DMA_APB_SA1	0x8808002C
44	APB DMA 第 2 通道起始地址	P_DMA_APB_SA2	0x88080030
45	APB DMA 第 3 通道起始地址	P_DMA_APB_SA3	0x88080034
46	AHB DMA 第 0 通道缓冲区 B 起始地址	P_DMA_AHB_SA0BB	0x8808004C
47	AHB DMA 第 1 通道缓冲区 B 起始地址	P_DMA_AHB_SA1BB	0x88080050
48	AHB DMA 第 2 通道缓冲区 B 起始地址	P_DMA_AHB_SA2BB	0x88080054
49	AHB DMA 第 3 通道缓冲区 B 起始地址	P_DMA_AHB_SA3BB	0x88080058
50	AHB DMA 第 0 通道缓冲区 B 结束地址	P_DMA_AHB_EA0BB	0x8808005C
51	AHB DMA 第 1 通道缓冲区 B 结束地址	P_DMA_AHB_EA1BB	0x88080060



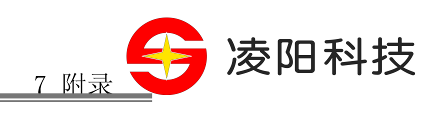
序号	寄存器名称	助记符	地址
52	AHB DMA 第 2 通道缓冲区 B 结束地址	P_DMA_AHB_EA2BB	0x88080064
53	AHB DMA 第 3 通道缓冲区 B 结束地址	P_DMA_AHB_EA3BB	0x88080068
54	DMA 第 0 通道控制寄存器	P_DMA_CHANNEL0_CTRL	0x8808006C
55	DMA 第 1 通道控制寄存器	P_DMA_CHANNEL1_CTRL	0x88080070
56	DMA 第 2 通道控制寄存器	P_DMA_CHANNEL2_CTRL	0x88080074
57	DMA 第 3 通道控制寄存器	P_DMA_CHANNEL3_CTRL	0x88080078
58	DMA 通道复位寄存器	P_DMA_CHANNEL_RESET	0x8808007C
GPIO			
59	GPIO 时钟配置寄存器	P_GPIO_CLK_CONF	0x882100FC
60	IOA 设置寄存器	P_IOA_GPIO_SETUP	0x88200038
61	IOB 设置寄存器	P_IOB_GPIO_SETUP	0x8820004C
62	IOB 输入数据寄存器	P_IOB_GPIO_INPUT	0x88200070
63	IOA 输入数据寄存器	P_IOA_GPIO_INPUT	0x88200074
64	IOA 外部中断寄存器	P_IOA_GPIO_INT	0x88200090
Timer 模块			
65	TIMER 时钟选择寄存器	P_TIMER_CLK_SEL	0x882100E4
66	TIMER 接口选择寄存器	P_TIMER_INTERFACE_SEL	0x88200010
67	TIMER0 时钟配置寄存器	P_TIMER0_CLK_CONF	0x8821006C
68	TIMER0 控制寄存器	P_TIMER0_MODE_CTRL	0x88160000
69	TIMER0 CCP 控制寄存器	P_TIMER0_CCP_CTRL	0x88160004
70	TIMER0 计数初值数据寄存器	P_TIMER0_PRELOAD_DATA	0x88160008
71	TIMER0 CCP 计数初值数据寄存器	P_TIMER0_CCP_DATA	0x8816000C
72	TIMER0 计数寄存器	P_TIMER0_COUNT_DATA	0x88160010
73	TIMER1 时钟配置寄存器	P_TIMER1_CLK_CONF	0x88210070
74	TIMER1 控制寄存器	P_TIMER1_MODE_CTRL	0x88161000
75	TIMER1 CCP 控制寄存器	P_TIMER1_CCP_CTRL	0x88161004
76	TIMER1 计数初值数据寄存器	P_TIMER1_PRELOAD_DATA	0x88161008
77	TIMER1 CCP 计数初值数据寄存器	P_TIMER1_CCP_DATA	0x8816100C
78	TIMER1 计数寄存器	P_TIMER1_COUNT_DATA	0x88161010

序号	寄存器名称	助记符	地址
79	TIMER2 时钟配置寄存器	P_TIMER2_CLK_CONF	0x88210074
80	TIMER2 控制寄存器	P_TIMER2_MODE_CTRL	0x88162000
81	TIMER2 CCP 控制寄存器	P_TIMER2_CCP_CTRL	0x88162004
82	TIMER2 计数初值数据寄存器	P_TIMER2_PRELOAD_DATA	0x88162008
83	TIMER2 CCP 计数初值数据寄存器	P_TIMER2_CCP_DATA	0x8816200C
84	TIMER2 计数寄存器	P_TIMER2_COUNT_DATA	0x88162010
85	TIMER3 时钟配置寄存器	P_TIMER3_CLK_CONF	0x88200078
86	TIMER3 控制寄存器	P_TIMER3_MODE_CTRL	0x88163000
87	TIMER3 CCP 控制寄存器	P_TIMER3_CCP_CTRL	0x88163004
88	TIMER3 计数初值数据寄存器	P_TIMER3_PRELOAD_DATA	0x88163008
89	TIMER3 CCP 计数初值数据寄存器	P_TIMER3_CCP_DATA	0x8816300C
90	TIMER3 计数寄存器	P_TIMER3_COUNT_DATA	0x88163010
91	TIMER4 时钟配置寄存器	P_TIMER4_CLK_CONF	0x8821007C
92	TIMER4 控制寄存器	P_TIMER4_MODE_CTRL	0x88164000
92	TIMER4 CCP 控制寄存器	P_TIMER4_CCP_CTRL	0x88164004
94	TIMER4 计数初值数据寄存器	P_TIMER4_PRELOAD_DATA	0x88164008
95	TIMER4 CCP 计数初值数据寄存器	P_TIMER4_CCP_DATA	0x8816400C
96	TIMER4 计数寄存器	P_TIMER4_COUNT_DATA	0x88164010
97	TIMER5 时钟配置寄存器	P_TIMER5_CLK_CONF	0x88210080
98	TIMER5 控制寄存器	P_TIMER5_MODE_CTRL	0x88165000
99	TIMER5 CCP 控制寄存器	P_TIMER5_CCP_CTRL	0x88165004
100	TIMER5 计数初值数据寄存器	P_TIMER5_PRELOAD_DATA	0x88165008
101	TIMER5 CCP 计数初值数据寄存器	P_TIMER5_CCP_DATA	0x8816500C
102	TIMER5 计数寄存器	P_TIMER5_COUNT_DATA	0x88165010
实时时钟模块			
103	RTC 时钟配置寄存器	P_RTC_CLK_CONF	0x88210088
104	RTC 秒寄存器	P_RTC_TIME_SEC	0x88166000
105	RTC 分寄存器	P_RTC_TIME_MIN	0x88166004
106	RTC 时寄存器	P_RTC_TIME_HOUR	0x88166008



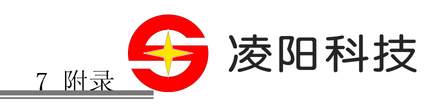
序号	寄存器名称	助记符	地址
107	RTC 秒报警寄存器	P_RTC_ALM_SEC	0x8816600C
108	RTC 分报警寄存器	P_RTC_ALM_MIN	0x88166010
109	RTC 时报警寄存器	P_RTC_ALM_HOUR	0x88166014
110	RTC 控制寄存器	P_RTC_MODE_CTRL	0x88166018
111	RTC 中断状态寄存器	P_RTC_INT_STATUS	0x8816601C
时基模块			
112	TMB 时钟配置寄存器	P_TMB_CLK_CONF	0x882100E0
113	TMB 控制寄存器	P_TMB_MODE_CTRL	0x88166020
114	TMB 中断状态寄存器	P_TMB_INT_STATUS	0x88166024
115	TMB 复位命令寄存器	P_TMB_RESET_COMMAND	0x88166028
WDOG 模块			
116	WDOG 时钟配置寄存器	P_WDOG_CLK_CONF	0x88210084
117	WDOG 复位状态寄存器	P_WDOG_RESET_STATUS	0x882100E8
118	WDOG 控制寄存器	P_WDOG_MODE_CTRL	0x88170000
119	WDOG 复位周期设置寄存器	P_WDOG_CYCLE_SETUP	0x88170004
120	WDOG 清狗命令寄存器	P_WDOG_CLR_COMMAND	0x88170008
ADC 模块			
121	ADC 时钟配置寄存器	P_ADC_CLK_CONF	0x882100AC
122	ADC 时钟选择寄存器	P_ADC_CLK_SEL	0x882100B0
123	ADC GPIO 设置寄存器	P_ADC_GPIO_SETUP	0x88200048
124	ADC GPIO 输入数据寄存器	P_ADC_GPIO_INPUT	0x88200078
125	ADC GPIO 外部中断寄存器	P_ADC_GPIO_INT	0x8820009C
126	ADC 模拟输入控制寄存器	P_ADC_AINPUT_CTRL	0x88200054
127	ADC 控制寄存器 1	P_ADC_MIC_CTRL1	0x881a0000
128	MIC 增益寄存器	P_ADC_MIC_GAIN	0x881a0004
129	ADC 采样时钟寄存器	P_ADC_SAMPLE_CLK	0x881a0008
130	ADC 采样保持寄存器	P_ADC_SAMPLE_HOLD	0x881a000c
131	ADC 控制寄存器 2	P_ADC_MIC_CTRL2	0x881a0010
132	ADC 中断状态寄存器	P_ADC_INT_STATUS	0x881a0014

序号	寄存器名称	助记符	地址
133	ADC 手动方式数据寄存器	P_ADC_MANUAL_DATA	0x881a0018
134	ADC 自动方式数据寄存器	P_ADC_AUTO_DATA	0x881a001c
135	MIC 转换数据寄存器	P_ADC_MIC_DATA	0x881a0020
UART 模块			
136	UART 时钟配置寄存器	P_UART_CLK_CONF	0x8821005C
137	UART 接口选择寄存器	P_UART_INTERFACE_SEL	0x88200000
138	UART GPIO 设置寄存器	P_UART_GPIO_SETUP	0x88200040
139	UART GPIO 输入数据寄存器	P_UART_GPIO_INPUT	0x88200074
140	UART GPIO 外部中断寄存器	P_UART_GPIO_INT	0x88200094
141	UART 控制寄存器	P_UART_MODE_CTRL	0x88150008
142	UART 波特率设置寄存器	P_UART_BAUDRATE_SETUP	0x8815000C
143	UART 收发状态寄存器	P_UART_TXRX_STATUS	0x88150010
144	UART 错误状态寄存器	P_UART_ERR_STATUS	0x88150004
145	UART 收发数据寄存器	P_UART_TXRX_DATA	0x88150000
146	UART 唤醒状态寄存器	P_UART_WAKEUP_STATUS	0x88210110
SPI 模块			
147	SPI 时钟配置寄存器	P_SPI_CLK_CONF	0x88210098
148	SPI 接口选择寄存器	P_SPI_INTERFACE_SEL	0x882000A4
149	SPI 控制寄存器	P_SPI_MODE_CTRL	0x88110000
150	SPI 发送状态寄存器	P_SPI_TX_STATUS	0x88110004
151	SPI 发送数据寄存器	P_SPI_TX_DATA	0x88110008
152	SPI 接收状态寄存器	P_SPI_RX_STATUS	0x8811000C
153	SPI 接收数据寄存器	P_SPI_RX_DATA	0x88110010
154	SPI 收发状态寄存器	P_SPI_TXRX_STATUS	0x88110014
I2C 模块			
155	I2C 时钟配置寄存器	P_I2C_CLK_CONF	0x88210094
156	I2C 接口选择寄存器	P_I2C_INTERFACE_SEL	0x88200004
157	I2C GPIO 设置寄存器	P_I2C_GPIO_SETUP	0x88200044
158	I2C GPIO 输入数据寄存器	P_I2C_GPIO_INPUT	0x88200074



序号	寄存器名称	助记符	地址
159	I2C GPIO 外部中断寄存器	P_I2C_GPIO_INT	0x88200098
160	I2C 控制寄存器	P_I2C_MODE_CTRL	0x88130020
161	I2C 中断状态寄存器	P_I2C_INT_STATUS	0x88130024
162	I2C 传输速率设置寄存器	P_I2C_RATE_SETUP	0x88130028
163	I2C 从机地址寄存器	P_I2C_SLAVE_ADDR	0x8813002C
164	I2C 数据地址寄存器	P_I2C_DATA_ADDR	0x88130030
165	I2C 发送数据寄存器	P_I2C_TX_DATA	0x88130034
166	I2C 接收数据寄存器	P_I2C_RX_DATA	0x88130038
SIO 模块			
167	SIO 接口选择寄存器	P_SIO_INTERFACE_SEL	0x88200004
168	JTAG GPIO 设置寄存器	P_JTAG_GPIO_SETUP	0x88200034
169	JTAG GPIO 输入数据寄存器	P_JTAG_GPIO_INPUT	0x88200070
170	JTAG GPIO 外部中断寄存器	P_JTAG_GPIO_INT	0x8820008C
171	SIO 时钟配置寄存器	P_SIO_CLK_CONF	0x882100A0
172	SIO 控制寄存器	P_SIO_MODE_CTRL	0X88120000
173	SIO 自动传输控制寄存器	P_SIO_AUTO_CTRL	0x88210004
174	SIO 数据地址寄存器	P_SIO_DATA_ADDR	0x88210008
175	SIO 收发数据寄存器	P_SIO_TXRX_DATA	0x8821000C
NAND Flash 模块			
176	NAND 接口选择寄存器	P_NAND_INTERFACE_SEL	0x882000A4
177	NAND GPIO 设置寄存器	P_NAND_GPIO_SETUP	0x8820002C
178	NAND GPIO 上下拉寄存器	P_NAND_GPIO_PULL	0x88200030
179	NAND GPIO 输入数据寄存器	P_NAND_GPIO_INPUT	0x8820006C
180	NAND GPIO 外部中断寄存器	P_NAND_GPIO_INT	0x88200088
181	NAND 时钟配置寄存器	P_NAND_CLK_CONF	0x882100A8
182	NAND 中断控制寄存器	P_NAND_INT_CTRL	0x88190014
183	NAND 控制寄存器	P_NAND_MODE_CTRL	0x88190000
184	NAND 命令寄存器	P_NAND_CLE_COMMAND	0x88190004
185	NAND 地址寄存器	P_NAND_ALE_ADDR	0x88190008

序号	寄存器名称	助记符	地址
186	NAND 发送数据寄存器	P_NAND_TX_DATA	0x8819000C
187	NAND 接收数据寄存器	P_NAND_RX_DATA	0x88190010
188	NAND 中断状态寄存器	P_NAND_INT_STATUS	0x88190018
189	NAND 真实行校验码寄存器	P_NAND_ECC_TRUELP	0x8819001C
190	NAND 真实列校验码寄存器	P_NAND_ECC_TRUECP	0x88190020
191	NAND 对比行校验码寄存器	P_NAND_ECC_CMPLP	0x88190024
192	NAND 对比列校验码寄存器	P_NAND_ECC_CMPCP	0x88190028
192	NAND 校验状态寄存器	P_NAND_ECC_STATUS	0x8819002C
194	NAND 校验控制寄存器	P_NAND_ECC_CTRL	0x88190030
SD Card 模块			
195	SD 接口选择寄存器	P_SD_INTERFACE_SEL	0x882000A4
196	SD 时钟配置寄存器	P_SD_CLK_CONF	0x882100A4
197	SD 控制寄存器	P_SD_MODE_CTRL	0x88180018
198	SD 中断控制寄存器	P_SD_INT_CTRL	0x8818001C
199	SD 中断状态寄存器	P_SD_INT_STATUS	0x88180014
200	SD 命令设置寄存器	P_SD_COMMAND_SETUP	0x88180008
201	SD 参数数据寄存器	P_SD_ARGUMENT_DATA	0x8818000C
202	SD 响应数据寄存器	P_SD_RESPONSE_DATA	0x88180010
203	SD 发送数据寄存器	P_SD_TX_DATA	0x88180000
204	SD 接收数据寄存器	P_SD_RX_DATA	0x88180004
TFT LCD 模块			
205	TFT 控制寄存器	P_TFT_MODE_CTRL	0x88040000
206	TFT 输出数据格式寄存器	P_TFT_DATA_FMT	0x88040004
207	TFT 水平显示扫描寄存器	P_TFT_HOR_ACT	0x88040008
208	TFT 水平空白前扫寄存器	P_TFT_HOR_FRONT	0x8804000C
209	TFT 水平空白后扫寄存器	P_TFT_HOR_BACK	0x88040010
210	TFT 水平同步扫描寄存器	P_TFT_HOR_SYNC	0x88040014
211	TFT 垂直显示扫描寄存器	P_TFT_VER_ACT	0x88040018
212	TFT 垂直空白前扫寄存器	P_TFT_VER_FRONT	0x8804001C



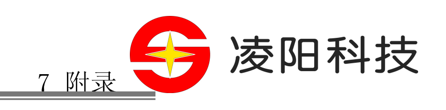
序号	寄存器名称	助记符	地址
213	TFT 垂直空白后扫寄存器	P_TFT_VER_BACK	0x88040020
214	TFT 垂直同步扫描寄存器	P_TFT_VER_SYNC	0x88040024
215	TFT 帧数据格式寄存器 1	P_TFT_FRAME_FMT1	0x88040028
216	TFT 起始行寄存器	P_TFT_ROW_START	0x8804002C
217	TFT 起始列寄存器	P_TFT_COL_START	0x88040030
218	TFT 列宽度寄存器	P_TFT_COL_WIDTH	0x88040034
219	TFT 余量宽度寄存器	P_TFT_DUMMY_WIDTH	0x88040038
220	TFT 状态寄存器	P_TFT_BUFFER_STATUS	0x8804003C
221	TFT 输出数据顺序寄存器	P_TFT_DATA_SEQ	0x88040040
222	TFT 中断状态寄存器	P_TFT_INT_STATUS	0x88040050
223	TFT 帧数据格式寄存器 2	P_TFT_FRAME_FMT2	0x880400A0
224	LCD 时钟配置寄存器	P_LCD_CLK_CONF	0x88210034
225	LCD 时钟选择寄存器	P_LCD_CLK_SEL	0x88210038
226	LCD 接口选择寄存器	P_LCD_INTERFACE_SEL	0x88200000
227	LCD 显示缓冲区起始地址寄存器 1	P_LCD_BUFFER_SA1	0x8807000C
228	LCD 显示缓冲区起始地址寄存器 2	P_LCD_BUFFER_SA2	0x88070010
229	LCD 显示缓冲区起始地址寄存器 3	P_LCD_BUFFER_SA3	0x88070014
230	LCD 显示缓冲区选择寄存器	P_LCD_BUFFER_SEL	0x88090024
231	TFT GPIO 输出数据寄存器	P_TFT_GPIO_DATA	0x88200014
232	TFT GPIO 输出使能寄存器	P_TFT_GPIO_OUTPUTEN	0x88200018
233	TFT GPIO 上拉寄存器	P_TFT_GPIO_PULLUP	0x8820001C
234	TFT GPIO 下拉寄存器	P_TFT_GPIO_PULLDOWN	0x88200020
235	TFT GPIO 输入数据寄存器	P_TFT_GPIO_INPUT	0x88200064
236	TFT GPIO 外部中断寄存器	P_TFT_GPIO_INT	0x88200080

7.4 汇编指令速查表

表 7.5 32 位指令速查表

序号	指令	功能说明	语法格式
1	lb	装载字节数据	lb rD, [rA, SImm15]

序号	指令	功能说明	语法格式
		装载字节数据(后变址寻址)	lb rD, [rA]+, SImm12
		装载字节数据(前变址寻址)	lb rD, [rA, SImm12]+
2	lbu	装载无符号字节数据	lbu rD, [rA, SImm15]
		装载无符号字节数据(后变址寻址)	lbu rD, [rA]+, SImm12
		装载无符号字节数据(前变址寻址)	lbu rD, [rA, SImm12]+
3	lh	装载半字数据	lh rD, [rA, SImm15]
		装载半字数据(后变址寻址)	lh rD, [rA]+, SImm12
		装载半字数据(前变址寻址)	lh rD, [rA, SImm12]+
4	lhu	装载无符号半字数据	lhu rD, [rA, SImm15]
		装载无符号半字数据(后变址寻址)	lhu rD, [rA]+, SImm12
		装载无符号半字数据(前变址寻址)	lhu rD, [rA, SImm12]+
5	lw	装载字数据	lw rD, [rA, SImm15]
		装载字数据(后变址寻址)	lw rD, [rA]+, SImm12
		装载字数据(前变址寻址)	lw rD, [rA, SImm12]+
6	sb	存储字节数据	sb rD, [rA, SImm15]
		存储字节数据(后变址寻址)	sb rD, [rA]+, SImm12
		存储字节数据(前变址寻址)	sb rD, [rA, SImm12]+
7	sh	存储半字数据	sh rD, [rA, SImm15]
		存储半字数据(后变址寻址)	sh rD, [rA]+, SImm12
		存储半字数据(前变址寻址)	sh rD, [rA, SImm12]+
8	sw	存储字数据	sw rD, [rA, SImm15]
		存储字数据(后变址寻址)	sw rD, [rA]+, SImm12
		存储字数据(前变址寻址)	sw rD, [rA, SImm12]+
9	ldi	装载立即数	ldi rD, SImm16
10	ldis	装载移位立即数	ldis rD, SImm16
11	lcb	装载合并字数据开始	lcb [rA]+
12	lcw	装载合并字数据	lcw rD, [rA]+
13	lce	装载合并字数据结束	lce rD, [rA]+
14	scb	存储合并字数据开始	scb rD, [rA]+



序号	指令	功能说明	语法格式
15	scw	存储合并字数据	scw rD, [rA]+
16	sce	存储合并字数据结束	sce [rA]+
17	add{.c}	加法运算	add{.c} rD, rA, rB
18	addc{.c}	带进位加法运算	addc{.c} rD, rA, rB
19	addi{.c}	立即数加法运算	addi{.c} rD, SImm16
20	addis{.c}	移位立即数加法运算	addis{.c} rD, SImm16
21	addri{.c}	寄存器立即数加法运算	addri{.c} rD, rA, SImm14
22	sub{.c}	减法运算	sub{.c} rD, rA, rB
23	subc{.c}	带借位减法运算	subc{.c} rD, rA, rB
24	subi{.c}	立即数减法运算	subi{.c} rD, SImm16
25	subis{.c}	移位立即数减法运算	subis{.c} rD, SImm16
26	subri{.c}	寄存器立即数减法运算	subri{.c} rD, rA, SImm14
27	neg{.c}	取负	neg{.c} rD, rB
28	cmp{TCS}.c	比较	cmp{TCS}.c rA, rB
29	cmpi.c	立即数比较	cmpi.c rA, SImm16
30	cmpz{TCS}.c	零比较	cmpz{TCS}.c rA
31	mul	乘法运算	mul rA, rB
32	mulu	无符号数乘法运算	mulu rA, rB
33	div	除法运算	div rA, rB
34	divu	无符号数除法运算	divu rA, rB
35	and{.c}	逻辑与	and{.c} rD, rA, rB
36	andi{.c}	立即数逻辑与	andi{.c} rD, Imm16
37	andis{.c}	移位立即数逻辑与	andis{.c} rD, Imm16
38	andri{.c}	寄存器立即数逻辑与	andri{.c} rD, rA, Imm14
39	or{.c}	逻辑或	or{.c} rD, rA, rB
40	ori{.c}	立即数逻辑或	ori{.c} rD, Imm16
41	oris{.c}	移位立即数逻辑或	oris{.c} rD, Imm16
42	orri{.c}	寄存器立即数逻辑或	orri{.c} rD, rA, Imm14
43	xor{.c}	逻辑异或	xor{.c} rD, rA, rB

序号	指令	功能说明	语法格式
44	not{.c}	逻辑异非	not{.c} rD, rA
45	t{cond}	测试并置 T 条件标志	t{cond}
46	bittst.c	位测试	bittst.c rD, BN
47	bitset.c	位置位	bitset.c rD, rA, BN
48	bitclr.c	位清零	bitclr.c rD, rA, BN
49	bittgl.c	位翻转	bittgl.c rD, rA, BN
50	mv{cond}	数据传送	mv{cond} rD, rA
51	mtcr	传送数据到控制寄存器	mtcr rD, Crn
52	mfcr	从控制寄存器传送数据	mfcr rD, Crn
53	rol{.c}	循环左移	rol{.c} rD, rA, rB
54	rolc.c	带进位循环左移	rolc.c rD, rA, rB
55	roli{.c}	立即数循环左移	roli{.c} rD, rA, Imm5
56	rolic.c	立即数带进位循环左移	rolic.c rD, rA, Imm5
57	ror{.c}	循环右移	ror{.c} rD, rA, rB
58	rorc.c	带进位循环右移	rorc.c rD, rA, rB
59	rori{.c}	立即数循环右移	rori{.c} rD, rA, Imm5
60	roric.c	立即数带进位循环右移	roric.c rD, rA, Imm5
61	sll{.c}	逻辑左移	sll{.c} rD, rA, rB
62	slli{.c}	立即数逻辑左移	slli{.c} rD, rA, Imm5
63	sra{.c}	算术右移	sra{.c} rD, rA, rB
64	srai{.c}	立即数算术右移	srai{.c} rD, rA, Imm5
65	srl{.c}	逻辑右移	srl{.c} rD, rA, rB
66	srli{.c}	立即数逻辑右移	srli{.c} rD, rA, Imm5
67	extsb{.c}	扩展有符号字节数据	extsb{.c} rD, rA
68	extzb{.c}	扩展无符号字节数据	extzb{.c} rD, rA
69	extsh{.c}	扩展有符号半字数据	extsh{.c} rD, rA
70	extzh{.c}	扩展无符号半字数据	extzh{.c} rD, rA
71	j	无条件跳转	j{l} label
72	b	条件分支	b{cond}{l} label



序号	指令	功能说明	语法格式
73	br	条件寄存器分支	br{cond}{l} rA
74	ceinst	Custom Engine 用户定义	ceinst func5, rA, rB, USD1, USD2
75	mtcex	传送数据到 Custom Engine 寄存器	mtcel rD mtceh rD mtcehl rD, rA
76	mfcex	从 Custom Engine 寄存器传送数据到通用寄存器	mfcel rD mfceh rD mfcehl rD, rA
77	mtsr	传送数据到特殊寄存器	mtsr rA, Srn
78	mfsr	从特殊寄存器传送数据到通用寄存器	mfsr rD, Srn
79	cache	cache 控制指令	cache cache_op, [rA, SImm15]
80	trap	条件陷阱	trap{cond} Software_Parameter(Imm5)
81	syscall	系统调用	syscall Software_Parameter(Imm15)
82	sleep	睡眠	sleep
83	sdbbp	软件 Debug 断点	sdbbp code(Imm5)
84	pflush	清空流水线	pflush
85	rte	从异常返回	rte
86	drte	从 debug 异常返回	drte
87	COPn	协处理器用户定义（n 为协处理器编号）	COPn cop_code20 或 COPn CrD, CrA, CrB, cop_code5
88	MTCn	传送数据到协处理器数据寄存器	MTCn rD, CrA
89	MTCCn	传送数据到协处理器控制寄存器	MTCCn rD, CrA
90	MFCn	从协处理器数据寄存器传送数据	MFCn rD, CrA
91	MFCCn	从协处理器控制寄存器传送数据	MFCCn rD, CrA
92	LDCn	装载数据到协处理器数据寄存器	LDCn CrA, [rD, SImm12]
93	STCn	从协处理器数据寄存器存储数据	STCn CrA, [rD, SImm12]

表 7.6 16 位指令速查表

序号	指令	功能说明	格式

序号	指令	功能说明	格式
1	lbu{p}!	装载无符号字节数据	lbu! rDg0, [rAg0]基址寻址 lbup! rDgo, Imm5 基址变址寻址
2	lh{p}!	装载半字数据	lh! rDg0, [rAg0]　基址寻址 lhp! rDg0, Imm6　基址变址寻址
3	lw{p}!	装载字数据	lw! rDg0, [rAg0]　基址寻址 lwp! rDg0, Imm7 基址变址寻址
4	sb{p}!	存储字节数据	sb! rDg0, [rAg0]　基址寻址 sbp! rDg0, Imm5　基址变址寻址
5	sh{p}!	存储半字数据	sh! rDg0, [rAg0]　基址寻址 shp! rDg0, Imm6　基址变址寻址
6	sw{p}!	存储字数据	sw! rDg0, [rAg0]　基址寻址 swp! rDg0, Imm7 基址变址寻址
7	ldiu!	装载无符号立即数	ldiu! rDg0, Imm8
8	push!	存储字数据（前减量变址寻址）	push! rDgh, [rAg0]
9	pop!	装载字数据（后增量变址寻址）	pop! rDgh, [rAg0]
10	add!	16 位加法运算	add! rDg0, rAg0
11	addc!	16 位带进位加法运算	addc! rDg0, rAg0
12	addei!	16 位立即数加法运算	addei! rDg0, Imm4
13	sub!	16 位减法运算	sub! rDg0, rAg0
14	subei!	16 位立即数减法运算	subei! rDg0, Imm4
15	neg!	16 位取负	neg! rDg0, rAg0
16	cmp!	16 位比较	cmp! rDg0, rAg0
17	and!	16 位逻辑与	and! rDg0, rAg0
18	or!	16 位逻辑或	or! rDg0, rAg0
19	xor!	16 位逻辑异或	xor! rDg0, rAg0
20	not!	16 位逻辑异非	not! rDg0, rAg0
21	t{cond}!	16 位测试并置 T 条件标志	t{cond}!
22	bittst!	16 位位测试	bittst! rDg0, BN(Imm5)
23	bitset!	位置位	bitset! rDg0, BN
24	bitclr.c	位清零	bitclr! rDg0, BN

序号	指令	功能说明	格式
25	bittgl.c	位翻转	bittgl! rDg0, BN
26	mv!	16 位寄存器数据传送(低组->低组)	mv! rDg0, rAg0
27	mlfh!	16 位寄存器数据传送(高组->低组)	mlfh! rDg0, rAg1
28	mhfl!	16 位寄存器数据传送(低组->高组)	mhfl! rDg1, rAg0
29	sll!	16 位逻辑左移	sll! rDg0, rAg0
30	slli!	16 位立即数逻辑左移	slli! rDg0, Imm5
31	sra!	16 位算术右移	sra! rDg0, rAg0
32	srl!	16 位逻辑右移	srl! rDg0, rAg0
33	srli!	16 位立即数逻辑右移	srli! rDg0, Imm5

7.5 伪指令速查表

表 7.7 伪指令速查表

序号	伪指令	功能说明	格式
1	.align	将位置计数器 PC 对齐到值为 alignment 的整数倍的地址	.align alignment[, [fill][, max]]
2	.ascii	插入字符串	.ascii strings
3	.asciz	插入包含有字符串结束标志的字符串	.asciz strings
5	.byte	插入字节数据	.byte expressions
7	. data	切换到.data 段	.data [subsection]
8	. equ	全局符号 symbol 的地址赋值为 expression	.equ symbol, expression
9	. extern	申明外部函数或者标号	.extern symbol
10	. global	申明标号为全局符号	.global symbol
11	. hword	插入半字数据	.hword expression
12	. include	引入其他文件到当前文件内	.include filename
13	. int	在未初始化的段里插入 4 个字节数据	.int expressions
14	. org	将当前段的位置计数器 PC 后移到新地址	.org new-lc [, fill]
15	. section	定义一个段	.section NAME [, "FLAGS"[, @TYPE[, @ENTSIZE]]]
16	. set	禁止编译器产生使用某寄存器带来的警告	.set symbol, expression
17	. space	从当前位置插入 size 个字节数据	.space size [, fill]

序号	伪指令	功能说明	格式
18	. text	切换到.text 段	. text
19	. word	插入字数据	.word expression